

NTP WG  
Internet-Draft  
Intended status: Standards Track  
Expires: 6 December 2021

H. Gerstung  
M. Rohde  
D. Arnold  
Meinberg  
4 June 2021

Network Time Security for the Unicast Mode of the Precision Time  
Protocol  
draft-gerstung-nts4uotp-03

Abstract

This memo specifies the application of Network Time Security, a mechanism for using Transport Layer Security (TLS) and Authenticated Encryption with Associated Data (AEAD) to provide cryptographic security for the unicast mode of the Precision Time Protocol.

It is based on the 'Network Time Security for the Network Time Protocol' document RFC8915 and re-uses most of its mechanisms for providing a secure and robust key exchange solution for unicast PTP. Due to the different modes of operation, additional steps are required to secure unicast PTP communication between the PTP clients and unicast PTP servers. In addition to defining the new record types and other required values to allow the utilization of the NTS key exchange sub protocol, there are a number of additional protocol enhancements and server-side requirements which are defined in this memo.

NOTE

This document is work in progress

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 6 December 2021.

#### Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

#### Table of Contents

1. Introduction . . . . .	3
1.1. Terminology . . . . .	3
2. Objectives . . . . .	3
3. Application of the NTS protocol to PTP . . . . .	4
3.1. Phase 1: NTS-KE Phase . . . . .	7
3.2. Phase 2: PTP Unicast Transmission Negotiation Phase . . . . .	7
3.3. Phase 3: PTP Unicast Packet Transmission Phase . . . . .	12
4. New NTS record types . . . . .	13
4.1. Cookie for Unicast PTP . . . . .	13
4.2. Unicast PTP Server Negotiation . . . . .	13
5. The PTP client Table . . . . .	14
6. The NTS_TLV . . . . .	14
7. IANA Considerations . . . . .	17
8. Security Considerations . . . . .	17
8.1. Threat Model . . . . .	17
8.2. General Security Features . . . . .	18
9. Delay Attacks . . . . .	19
10. Acknowledgements . . . . .	19
11. References . . . . .	19
11.1. Normative References . . . . .	19
11.2. Informative References . . . . .	20
Authors' Addresses . . . . .	20

## 1. Introduction

This memo specifies Network Time Security for unicast mode of the Precision Time Protocol (PTP). It is based on [RFC8915] and applies the key exchange mechanism described there to PTP. The Precision Time Protocol is standardized in [IEEE1588] and offers a number of different modes and mappings to communication protocols. The security mechanisms described here provide a way to secure the unicast mode of PTP as specified in sub clause 16.1 of [IEEE1588].

The PTP integrated security mechanism has been specified in sub clause 16.14 of [IEEE1588] and introduces an AUTHENTICATION TLV that carries all necessary information to enable the receiver of a PTP message to verify its integrity. Although two different approaches are described in that sub clause (immediate and delayed security processing), NTS4UPTP only uses the immediate security processing.

In addition to sub clause 16.14, Annex P of [IEEE1588] provides additional explanation and description of PTP security. It is stated there that for key management it is assumed that a separate mechanism outside the context of PTP is used. In P.2.1.2 the document clearly states that this assumption was made in relation to the security mechanism described in 16.14 and it goes on to discuss some Key management options and it names both manual and automatic key management as possible approaches.

This memo describes a way to use the automatic key exchange mechanism as defined in [RFC8915] as the key management for unicast PTP. The NTS-KE protocol has clearly been designed to support using it for multiple time synchronization protocols and this document is utilizing this support.

### 1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 2. Objectives

The objectives of NTS are defined in [RFC8915] and, with some exceptions, apply to NTS4UPTP as well.

- \* Identity: Through the use of a X.509 public key infrastructure, implementations can cryptographically establish the identity of the parties they are communicating with.

- \* Authentication: Implementations can cryptographically verify that any time synchronization packets are authentic, i.e., that they were produced by an identified party and have not been modified in transit.
- \* Replay prevention: clients and servers can detect when a received packet is a replay of a previous packet.
- \* Request-response consistency: clients can verify that a unicast PTP packet received from a server was sent in response to a particular request from the client.
- \* Non-amplification: Implementations (especially server implementations) can avoid acting as distributed denial-of-service (DDoS) amplifiers by never responding to a packet with one or more packets creating more traffic than the initiating packet.
- \* Scalability: Server implementations can serve large numbers of clients.
- \* Performance: NTS must not significantly degrade the quality of the time transfer. The encryption and authentication used when actually transferring time should be lightweight.

The following objectives of [RFC8915] are not met in this proposal:

- \* Confidentiality: Basic time synchronization data is considered nonconfidential and sent in the clear. Despite this, NTS4NTP includes support for encrypting NTP extension fields. NTS4UPTP does not offer this kind of support as it is not considered useful or required for unicast PTP implementations.
- \* Unlinkability: For mobile clients, NTS4NTP does not leak any information additional to NTP which would permit a passive adversary to determine that two packets sent over different networks came from the same client. This objection cannot be achieved by unicast PTP because the protocol requires the server to keep a state for all its clients. It is also not considered a requirement in most applications where unicast PTP is deployed.

### 3. Application of the NTS protocol to PTP

Unlike NTP [RFC5905] which uses a request-response communication approach, the unicast mode of PTP is applying a subscription based model. Although [IEEE1588] allows unicast operation without negotiation this is rarely used. For this reason only unicast PTP with negotiation is considered. A PTP client (PTP Ordinary Clock, or synchronizing port of a PTP unicast Boundary Clock) sends a request

for the transmission of packets to a PTP instance offering such a service. This could be a PTP unicast Grandmaster instance or a PTP boundary clock. Note that [IEEE1588] allows PTP Ports in a states other than master to accept unicast message grant requests and act as a unicast PTP master. This option can be used for monitoring purposes. In this memo we treat all unicast associations in the same way regardless of whether it is for purposes of time transfer or monitoring, and refer to the port that grants message contracts as a PTP server.

A PTP server that receives a message grant request will then either accept the request or deny it, for example based on capacity considerations or its own operational state. This sub protocol of IEEE 1588 is called unicast message negotiation, an optional feature defined in sub clause 16.1 of [IEEE1588]. Both the PTP client and the PTP server granting a request can cancel a subscription (referred to as contract in PTP) after it has been granted and each contract includes a duration after which the PTP instance stops sending packets automatically if the PTP client did not request a new contract before the old contract ended.

This results in a 3-phase approach for PTP:

- \* Phase 1: NTS-KE Phase (Section 3.1)
- \* Phase 2: PTP Unicast Transmission Negotiation Phase (Section 3.2)
- \* Phase 3: PTP Unicast Packet Transmission Phase (Section 3.3)

In a typical use-case, phase 1 is required to be performed at startup. In phase 2 the PTP client and the PTP server will negotiate the transmission of PTP messages which will then be delivered by the PTP server in phase 3. Whenever phase 3 ends, the PTP client must re-run phase 2 to re-request more packets. Typically, PTP clients will re-run phase 2 before the active contract ends, i.e. they request a new transmission contract from the PTP server with a new duration before the active contract expires in order to secure a continuous flow of messages from the PTP server.

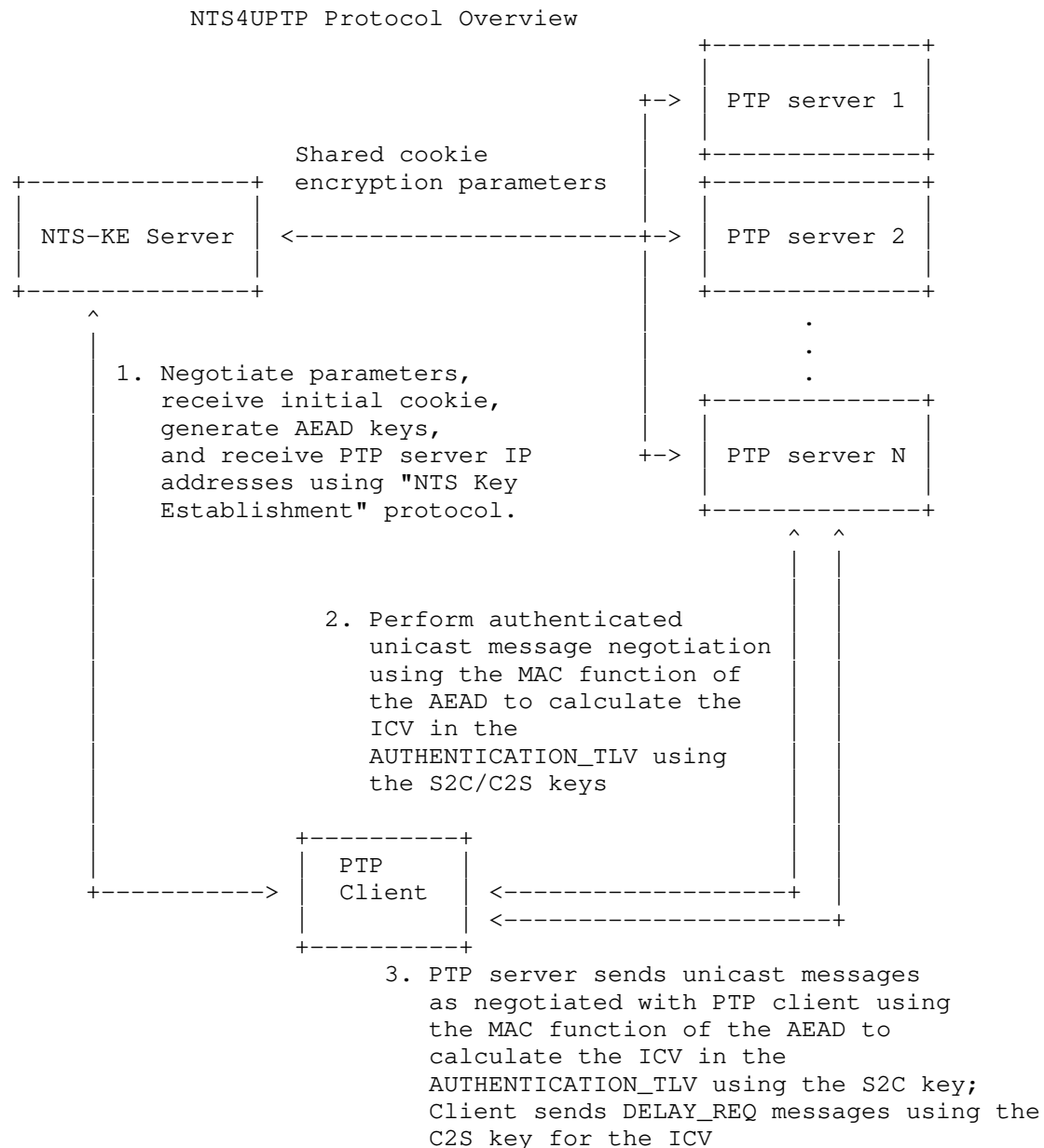


Figure 1: Overview of High-Level Interactions in NTS4UPTP

Phase 1 only needs to be re-run to avoid that the key lifetime expires, the PTP server stops responding or does not accept the cookie presented by the PTP client for any reasons.

### 3.1. Phase 1: NTS-KE Phase

In the NTS-KE Phase (Phase 1), the PTP client connects to the NTS-KE server via a secure TLS channel. Two keys are generated based on the TLS data exchange, referred to as Server-to-Client key (S2C key) and Client-to-Server key (C2S key). The NTS-KE server creates a cookie for the PTP client, which contains those two keys, the AEAD algorithm used and a Nonce. The cookie is secured by encrypting this information with a master key K. The master key is generated by a PTP server and sent to the KE server via a proprietary mechanism. The client therefore cannot decrypt a cookie as it does not know the master key. After receiving the cookies and establishing the C2S and S2C keys, the TLS connection is closed. This is identical to the NTS-KE phase as defined in [RFC8915] with the exception that the NTS-KE record type "next-protocol" points to PTP instead of NTP and two new NTS record types are introduced:

- \* "Cookie for Unicast PTP" provides the client with the cookie required to establish a valid NTS connection with the PTP server. This NTS record type is defined in Section 4.1.
- \* "PTP server Negotiation" tells the client which PTP server it MUST use. This NTS record type is defined in Section 4.2.

The MAC function of the AEAD algorithm will be used to create/calculate the ICV in the AUTHENTICATION\_TLV as defined in subclause 16.14 of [IEEE1588] (Integrated PTP Security).

For providing the cookie, an NTS-KE server MUST use a new NTS record type (New Cookie for Unicast PTP), which is identical to the NTS record type 5 (New Cookie for NTPv4) as described in [RFC8915].

The S2C key will be used in Phase 2 and 3 to calculate the ICV of the AUTHENTICATION\_TLV for all PTP messages sent from the PTP server to the PTP client. The C2S key will be used in Phase 2 and 3 to calculate the ICV of the AUTHENTICATION\_TLV for all PTP messages sent from the PTP client to the PTP server.

After the PTP client successfully completed Phase 1, it can enter Phase 2 by initiating the PTP unicast negotiation with the provided PTP server.

### 3.2. Phase 2: PTP Unicast Transmission Negotiation Phase

A unicast PTP client needs to establish a contract with a PTP server if it wants to receive SYNC and ANNOUNCE messages and to perform delay measurements by sending DELAY\_REQ messages to the PTP server, which responds with DELAY\_RESP messages.

The mechanism to establish these contracts between PTP client and PTP server is described in subclause 16.1 of [IEEE1588] ("Unicast message negotiation"). The basic concept requires the PTP client to send a request for each specific message type to transmit that message at a specific rate for a specific duration. The PTP server either grants the request or rejects it (for example due to capacity constraints). Each message type requires its own contract between a PTP client and a PTP server and there can only be one active contract per message type between the two nodes.

According to [IEEE1588] PTP messages can be extended by adding one or more TLVs on the end of the PTP message, and [IEEE1588] defines a number of TLVs. Here TLV stands for type-length-value. A standards development organization can also define TLVs to support specifications for extending PTP. In this case the TLV will be "ORGANIZATION\_EXTENSION\_PROPAGATE" or "ORGANIZATION\_EXTENSION\_DO\_NOT\_PROPAGATE" depending on whether a PTP Boundary Clock shall pass the TLV on or not. This kind of TLV MUST include a field for with the organizationID and a field with the organizationSubType. The latter MUST be unique among PTP TLVs defined by that organization.

The nature of unicast PTP requires that a PTP server maintains a list of PTP clients with active contracts. For NTS4UPTP, a server needs to store additional data. The storage entity required for storing the additional data is referred to as the PTPCLTABLE (Section 5) in this document.

In order to secure the PTP unicast transmission negotiation, PTP clients and servers use cookies and nonces, and protect the integrity of the PTP signaling messages with the integrated security mechanism based on the AUTHENTICATION\_TLV described in [IEEE1588]. A PTP client initially requires a cookie for the first message it sends during this phase and will receive a nonce each time the PTP server sends a response. The initial cookie for the first request of the client is provided by the NTS-KE server in the NTS-KE Phase. For each consecutive message the PTP client sends, it MUST use the nonce received in the previous message from the PTP server and send that one back in the NONCE field of the NTS\_TLV.

Due to the fact, that the S2C/C2S key pair expires, the client is forced to get new keys from the NTS-KE server. The client MAY do this at any time and MUST do it when receiving a NTS\_INVALIDKEYS response from the PTP server.

Nonces and cookies are not required in the third phase, in which the transmitted PTP messages are only secured with the AUTHENTICATION\_TLV. Packet rates in this phase can be very high, PTP

for example allows for up to 128 SYNC packets per second sent by the PTP server to a client. Due to the fact that PTP requires a client to successfully complete the negotiation phase, it is sufficient to protect the integrity of the messages in the transmission phase with the AUTHENTICATION\_TLV.

The unicast negotiation mechanism as specified in [IEEE1588] is carried out according to the standard, but with the following addition:

- \* The PTP client and the PTP server MUST add an NTS TLV and an AUTHENTICATION\_TLV to all signaling messages
- \* The NTS\_TLV (Section 6) in a message sent from the PTP client to the PTP server either carries the cookie that the client obtained during the last successfully completed NTS-KE Phase, or the last nonce provided by the PTP server in its response. Additionally the ntsMsgId MUST be set to NTS\_INIT for the first message sent to the PTP server after completing the NTS-KE phase and to NTS\_REQUEST for all further messages.
- \* The AUTHENTICATION\_TLV secures the REQUEST\_UNICAST\_TRANSMISSION\_TLV and the NTS\_TLV with an ICV which is calculated based on the MAC function of the AEAD algorithm and the C2S key that has been obtained during the NTS-KE phase
- \* All signaling messages from the PTP server to the PTP client MUST carry a new nonce and the ICV in the AUTHENTICATION\_TLV is calculated based on the S2C key that the PTP server read from the decrypted cookie in the last NTS\_INIT message from the client.

All PTP messages in the packet negotiation phase that do not carry an AUTHENTICATION\_TLV or in which the ICV is not correct MUST be ignored by the recipient.

A PTPCLTABLE entry SHALL be stored at least for as long as the S2C/C2S key pair is valid, i.e. until KEY\_PAIR\_EXP has been reached. The maximum lifetime of a key pair is defined in the configuration of the PTP server and the expiration date/time is calculated when the entry in this PTP client state storage is created (Expiration date/time=creation time of PTPCLTABLE record plus configured maximum lifetime). The PTP server SHOULD erase entries from this table after the expiration time of the key pair has been reached.

The PTP server, after receiving a signaling message, will perform the following steps:

- \* if the request contains an NTS\_TLV with ntsMsgId == NTS\_INIT, it decrypts the cookie with its master key K to obtain the two C2S and S2C keys, for all other ntsMsgId values it MUST perform a lookup into the PTPCLTABLE
- \* if the request contains an NTS\_TLV with ntsMsgId == NTS\_REQUEST, the nonce used in this signaling message is identical to the NEXT\_NONCE stored for this client in the PTPCLTABLE
- \* if the request contains an NTS\_TLV with ntsMsgId == NTS\_CHALLENGE\_REQUEST, CHALLENGE\_EXP has not been reached and the cookie used in this signaling message is identical to the CHALLENGE\_NONCE stored for this client
- \* If either the cookie cannot be decrypted, no matching entry could be found in the PTPCLTABLE, the nonce does not match the NEXT\_NONCE or the CHALLENGE\_NONCE, the message MUST be ignored. In all other cases, the following additional checks MUST be performed:
  - the ICV in the AUTHENTICATION\_TLV is correct (using the C2S key from the provided cookie or from the matching entry in the PTPCLTABLE)
  - the key expiration date/time has not been reached

If the CHALLENGE\_NONCE check fails (only applicable when the ntsMsgId in the received message is NTS\_CHALLENGE\_RESPONSE), the message MUST be ignored.

If the key expiration check fails, the request is denied and, by setting the key lifetime field of the NTS\_TLV in the response to 0 and the ntsMsgId to NTS\_INVALIDKEYS, the client is told to obtain new keys and a new cookie from a NTS-KE server before it can establish a new contract with this PTP server.

If no matching entry exists in the PTPCLTABLE, or if the checks above result require an NTS\_CHALLENGE\_REQUEST response, any active contract (if there is one) MUST NOT be changed, canceled or otherwise modified. This is to avoid that an attacker sends an invalid request which stops the currently active contract and therefore successfully carries out a denial-of-service attack.

When a PTP server receives a NTS\_INIT message with a valid cookie, the following challenge/response procedure MUST be followed:

- \* The PTP server will deny the REQUEST\_UNICAST\_TRANSMISSION\_TLV request and responds with an NTS\_MessageId NTS\_CHALLENGE\_REQUEST and a nonce which it stores as the CHALLENGE\_NONCE in the PTPCLTABLE. The server will put the sequenceId from the PTP packet header of the request into the reqSeqId field of the NTS\_TLV
- \* The client, after receiving the response with the NTS\_CHALLENGE\_REQUEST message ID SHALL check that the reqSeqId is identical to the sequenceId of the request it sent to the server. If this fails, the message MUST be ignored by the client.
- \* If the reqSeqId check is successful, the client resends the REQUEST\_UNICAST\_TRANSMISSION\_TLV using a ntsMsgId of NTS\_CHALLENGE\_RESPONSE and MUST use the cookie it received with the NTS\_CHALLENGE\_REQUEST response.
- \* The server then checks that the cookie presented by the client in the NTS\_CHALLENGE\_RESPONSE response is identical to the CHALLENGE\_NONCE. If that is true, the client can be trusted and the REQUEST\_UNICAST\_TRANSMISSION\_TLV included in this response is considered to be a valid and trusted request.

After the successful verification of the request, the REQUEST\_UNICAST\_TRANSMISSION\_TLV is processed according to [IEEE1588].

In case the PTP server grants the request to the client, the process moves on to the 3rd phase, the packet transmission phase. If the PTP server denies the request for whatever reason (i.e. it has no capacity or its state does not allow to reliably transmit the packets at the requested rate), it sends a unicast grant denial message, i.e. a PTP signaling message carrying a GRANT\_UNICAST\_TRANSMISSION\_TLV with the grant duration set to zero.

If a PTP client receives a GRANT\_UNICAST\_TRANSMISSION\_TLV containing an NTS\_TLV with a correct ICV and a key lifetime set to 0, it MUST delete all cookies it holds for that PTP server as well as the S2C/C2S key pair and cease all communication until it re-ran phase 1 and obtained a new key pair and a new cookie.

Using the "maximum key lifetime" configuration parameter, a PTP server operator can prioritize memory requirements and required network traffic volume. A small maximum lifetime value results in PTPCLTABLE entries being deleted earlier, requiring more NTS\_CHALLENGE\_REQUEST exchanges between PTP client and PTP server. A large value may result in requiring fewer such packet exchanges but increases the memory consumption because PTPCLTABLE entries have to be stored for a longer period of time.

### 3.3. Phase 3: PTP Unicast Packet Transmission Phase

When the transmission request has been granted by the PTP server for a specific messagetype/duration, for all messagetypes except delay responses the GM immediately starts transmitting the messages in the requested rate. DELAY\_RESP messages will be sent after the client sent a DELAY\_REQ message.

All unicast PTP messages sent by the PTP server to the PTP client due to an active contract SHALL be secured by an AUTHENTICATION\_TLV that carries an ICV as described in [IEEE1588], subclause 16.14 (PTP Integrated Security). The ICV is created using the MAC function of the AEAD algorithm and the S2C key established between the PTP client and the NTS-KE server during the NTS-KE phase. All messages sent by the PTP client to the PTP server will be secured with the same mechanism, but using the C2S key.

All PTP messages in the packet transmission phase that do not carry an AUTHENTICATION\_TLV or in which the ICV is not correct MUST be ignored by the recipient.

In order to establish a protection against replay attacks in this phase, both the PTP client and the PTP server MUST check that the sequenceId of an incoming message is larger than the sequenceId of the same PTP message type received in the previous message. If an incoming message does not pass the sequenceId check, it MUST be ignored. Both PTP client and PTP server SHOULD allow to configure the maximum difference between the sequenceId values of two consecutive messages of the same message type. They MUST gracefully handle the rollover of a sequenceId, which is a unsigned int16 value (0-65535).

To avoid that an attacker resends a PTP message with a sequenceId that has been obtained before the last rollover, additional integrity checks SHOULD be applied. The maximum packet rate is 128 packets/second. Therefore, for each PTP message type sent at the maximum rate, the sequenceId rollover happens every 512 seconds (8 minutes, 32 seconds) as a minimum. For SYNC, DELAY\_REQ and ANNOUNCE messages the recipient SHOULD check that the originTimestamp in the packet

does not differ more than 8 minutes from the originTimestamp of the previously received message. For DELAY\_RESP messages, the receiveTimestamp SHOULD be used instead.

The packet transmission phase ends either when the contract expired or when either the PTP server or the PTP client cancels the contract.

#### 4. New NTS record types

##### 4.1. Cookie for Unicast PTP

The content of this NTS record is identical to record type 5 as defined in 4.1.6 of [RFC8915]. However, a NTS-KE server MUST send exactly one record of this type when PTP is negotiated as a next protocol.

##### 4.2. Unicast PTP Server Negotiation

The PTP server Negotiation record has a Record Type number of 8. Its body consists of an ASCII-encoded [RFC0020] string. The contents of the string SHALL be either an IPv4 address, an IPv6 address, or a fully qualified domain name (FQDN). IPv4 addresses MUST be in dotted decimal notation. IPv6 addresses MUST conform to the "Text Representation of Addresses" as specified in RFC 4291 [RFC4291] and MUST NOT include [RFC6874]. If a label contains at least one non-ASCII character, it is an internationalized domain name, and an A-LABEL MUST be used as defined in Section 2.3.2.1 of RFC 5890 [RFC5890]. If the record contains a domain name, the recipient MUST treat it as a FQDN, e.g., by making sure it ends with a dot.

When PTP is negotiated as a Next Protocol and this record is sent by the server, the body specifies the hostname or IP address of the PTP unicast server with which the client MUST associate and that will accept the supplied cookies. If no record of this type is sent, the client SHALL interpret this as a directive to associate with a PTP server at the same IP address as the NTS-KE server. Servers MUST NOT send more than one record of this type. If the record contains a FQDN which resolves to multiple addresses, the client MUST choose at least one of the addresses the FQDN resolves to. The client MAY choose to use more than one address to request synchronization services from multiple unicast PTP servers in parallel.

When this record is sent by the client, it indicates that the client wishes to associate with the specified PTP server. The NTS-KE server MAY incorporate this request when deciding which PTP server Negotiation records to respond with, but honoring the client's preference is OPTIONAL. The client MUST NOT send more than one record of this type.

If the client has sent a record of this type, the NTS-KE server SHOULD reply with the same record if it is valid and the server is able to supply cookies for it. If the client has not sent any record of this type, the NTS-KE server SHOULD respond with either a PTP server address in the same family as the NTS-KE session or a FQDN that can be resolved to an address in that family, if such alternatives are available.

Servers MAY set the Critical Bit on records of this type; clients SHOULD NOT.

#### 5. The PTP client Table

The PTP server MUST store the following data for each PTP client in a NTS PTP client Table (PTPCLTABLE):

- \* the S2C/C2S key pair
- \* the time when the validity of this key pair expires (KEY\_PAIR\_EXP)
- \* the next nonce to be expected from the client (NEXT\_NONCE)
- \* the nonce to be expected from the client in a NTS\_CHALLENGE\_RESPONSE message (CHALLENGE\_NONCE)
- \* the time when the validity of the CHALLENGE\_NONCE expires (CHALLENGE\_EXP)

#### 6. The NTS\_TLV

The NTS\_TLV contains:

1. tlvType: ORGANIZATION\_EXTENSION\_DO\_NOT\_PROPAGATE (8000 hex)
2. lengthField: number of octets in the value + 6
3. organizationID: IETF (=00005E hex)
4. organizationSubtype: TBD (needs to be assigned)
5. ntsMsgId: NTS Message Id (see below)
6. networkProtocol: Transport type (0x01=IPv4, 0x02=IPv6, 0x03=Ethernet), unsigned int
7. tSrcAddr: Transport source address of the sender, e.g. the IP address or Ethernet MAC address, 16 octets

8. keyLifetime: Key Lifetime in seconds, unsigned int32
9. reqSeqId: sequenceId of the request, unsigned int16
10. nonce/cookie: a nonce or, if ntsMsgId == NTS\_INIT, an NTS Cookie

Bits								Octets	TLV Offset
0	1	2	3	4	5	6	7		
tlvType								2	0
lengthField								2	2
organizationId								3	4
organizationSubType								3	7
ntsMsgId								1	10
networkProtocol								1	11
tSrcAddr								16	12
keyLifetime								4	28
reqSeqId								2	32
nonce/cookie								n	34

Figure 2: NTS TLV Format

The networkProtocol field allows to detect which transport protocol is in use and therefore how to interpret the tSrcAddr field. For an Ethernet MAC address, the 6 first octets of the tSrcAddr field are relevant, for an IPv4 address the first 4 octets are relevant and for an IPv6 address the full 16 octets are relevant. The enumeration is corresponding to the networkProtocol field as defined in [IEEE1588].

Please note that the size of the cookie depends on the chosen AEAD, it can therefore differ and has been placed at the end of the TLV. The lengthField allows a receipt to calculate the length of the cookie.

The ntsMsgId field MUST contain one of the following values:

- \* 0x01 NTS\_INIT (for the first unicast negotiation message sent by the PTP client to the PTP server after completing a NTS-KE Phase in which the PTP client obtained a new key pair)
- \* 0x02 NTS\_REQUEST (for unicast negotiation messages sent by the PTP client to the PTP server)
- \* 0x03 NTS\_RESPONSE (for unicast negotiation messages sent by the PTP server to the PTP client)
- \* 0x04 NTS\_CHALLENGE\_REQUEST (for messages sent by the PTP server to the PTP client in case of a challenge/response procedure)
- \* 0x05 NTS\_CHALLENGE\_RESPONSE (for messages sent by the PTP client to the PTP server as a response to a NTS\_CHALLENGE\_REQUEST)
- \* 0x06 NTS\_INVALIDKEYS (for messages sent by the PTP server to the PTP client when the S2C/C2S key pair expired or whenever the PTP server wants to force the PTP client to re-run the NTS-KE Phase)

The reqSeqId field MUST be set to 0 for all messages except those with a NTS\_CHALLENGE\_REQUEST message Id. In that specific case the field MUST contain the sequenceId of the message to which this NTS\_CHALLENGE\_REQUEST is a response.

When sending a REQUEST\_UNICAST\_TRANSMISSION\_TLV, the PTP client will add the NTS\_TLV containing a cookie for this PTP server. The key lifetime SHALL always be set to 0 for all communication from the PTP client to the PTP server.

When sending a GRANT\_UNICAST\_TRANSMISSION\_TLV, the PTP server will add the NTS\_TLV as well. If the request of the client is granted (duration > 0), the NTS\_TLV will contain a new cookie and the key lifetime field SHALL represent the number of seconds the C2S and S2C keys continue to be valid.

The PTP server SHALL set a key lifetime of 0 when the lifetime of the S2C/C2S key pair expired. This allows the PTP server to force the client to re-start and obtain fresh keys and cookies from the NTS-KE server. When sending an NTS\_TLV with the key lifetime set to 0, the PTP server SHALL use the S2C key of the expired key pair to form the ICV in the AUTHENTICATION\_TLV, allowing the client to verify that the message has been sent by the PTP server.

When a client receives any message with a valid ICV but a key lifetime of 0 in the NTS\_TLV, it SHALL delete all cookies cease to send any messages to the PTP server and only restores communication with it after it obtained a new S2C/C2S key pair and a new set of cookies from the NTS-KE server.

## 7. IANA Considerations

RFC EDITOR: A new entry for Unicast PTP is required in the IANA Network Time Security Next Protocols Registry. The authors propose to add an entry with the Protocol ID = 1, the Protocol Name = "Unicast Precision Time Protocol" and a Reference to this draft. Please remove this comment before publishing and replace it with the data for the newly created entry from IANA.

## 8. Security Considerations

### 8.1. Threat Model

The procedures and mechanisms described in this draft protect against the following scenarios:

- \* Man-in-the-Middle (MITM) attack: All PTP nodes can verify that PTP messages received from another PTP server have not been modified in transit by an attacker. This is achieved by verifying that the ICV in the AUTHENTICATION\_TLV of every PTP message received is valid and has been created using the MAC function of the AEAD algorithm and the C2S or S2C key as established during the NTS-KE phase.
- \* Phase 2 Replay Attack (resending unmodified PTP unicast negotiation messages): PTP message integrity can be secured with the AUTHENTICATION\_TLV. However, with PTP this mechanism does not protect the transport protocol header and could therefore be used by an attacker to intercept a PTP message and then resending it using the same or a different source address. Or it could be resend at a later time to repeat a request or cancel an active contract. Resending it with the same address can be used to try and establish a contract for a unicast PTP client that is no longer requiring the service, requires the service in a different form (e.g. different message rates) or it can be used to cancel a contract (when resending a CANCEL\_UNICAST\_TRANSMISSION\_TLV after the client established a new contract). This can interrupt a required service for a PTP client or result in the PTP server unnecessarily consuming resources. By storing the nonce that has been provided by the server and that MUST be used by the client in its next request (NEXT\_NONCE in PTPCLTABLE), an unmodified resent REQUEST\_UNICAST\_TRANSMISSION\_TLV will not be granted, despite the

fact that the ICV in the AUTHENTICATION\_TLV is valid. To provide a robust defense against replaying NTS\_INIT messages, the challenge/response mechanism (NTS\_CHALLENGE\_REQUEST/NTS\_CHALLENGE\_RESPONSE) will require a PTP client to apply the correct C2S key and therefore protects against replaying a previously sent valid message with a cookie that has been encrypted with a still valid master key K.

- \* Phase 3 Replay Attack (resending unmodified PTP unicast messages): In Phase 3 the PTP server is sending PTP SYNC and ANNOUNCE messages to the PTP client and the PTP client is sending DELAY\_REQ messages to the PTP server, which responds with DELAY\_RESP messages. An attacker could resend any of these packets. For SYNC messages, that would result in the PTP client receiving "old time", i.e. the timestamps in such a message would be outdated and could disrupt time synchronization of the PTP client. A resent ANNOUNCE message could carry outdated information and therefore could force the PTP client to drop this server as a valid time source or distrust the protocol in other ways. Resending DELAY\_REQ messages could consume resources on the PTP server and, if the server would send DELAY\_RESP message, on the PTP client as well. The client could also be negatively affected by resent DELAY\_RESP messages that carry outdated time. A valid protection against such an attack is checking the sequenceId of each incoming message and by applying additional integrity checks to messages passing the sequenceId checks. See Section 3.3 for a detailed description of how this can be achieved.
- \* Amplification Attack/Distributed Denial of Service: Resending a REQUEST\_UNICAST\_TRANSMISSION\_TLV with a different source IP address could result in the PTP server sending PTP messages to IP addresses that do not expect and require receiving these messages. Due to the nature of unicast PTP, the traffic amplification potential is very high because one PTP signaling message containing a REQUEST\_UNICAST\_TRANSMISSION\_TLV can generate thousands of PTP messages from the PTP server to the source IP address used in the request message. This is mitigated by including the IP address of the originator of a message as a field (tSrcAddr) in the NTS\_TLV. The TLV as part of the PTP message is protected by the ICV in the AUTHENTICATION\_TLV and therefore a modified source address can be detected.

## 8.2. General Security Features

In addition to the above outlined protection mechanisms against specific attack scenarios, this draft also includes a generic security feature:

- \* **Key Freshness:** The expiration of C2S/S2C key pairs requires clients to obtain a new key pair in a configurable interval, which limits the time an attacker has to break those keys.

## 9. Delay Attacks

If an attacker gains control over a part of the network infrastructure on the path between clients and server, it could be possible to delay the forwarding of unicast PTP messages without modifying them. Applying such a delay only in one direction (e.g. for SYNC packets sent from the server to the client) would create an asymmetry in the delay calculation and as a result the error in the delay calculation would cause a timing error on the client. This kind of attack cannot be mitigated by NTS4UPTP as the cryptographic protection only allows to ensure the integrity of messages, which is not corrupted by a delay attack.

In addition to securing the network infrastructure (i.e. routers and switches) against this threat, another possible mitigation strategy for the client is to check the calculated delay against a static limit (which could be configurable by the user or is defined by the requirements of the application) or a dynamic limit, which the client could determine periodically for example by applying suitable statistical methods to determine a change in the calculated delay that indicates that the potential time error would exceed the sync requirements of the application.

## 10. Acknowledgements

The authors would like to thank the following contributors for their valuable feedback:

- \* Martin Langer, MSc and Prof. Dr.-Ing. Rainer Bermbach, Ostfalia University

## 11. References

### 11.1. Normative References

- [RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <<https://www.rfc-editor.org/info/rfc20>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<https://www.rfc-editor.org/info/rfc4291>>.
- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.
- [RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <<https://www.rfc-editor.org/info/rfc8915>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<https://www.rfc-editor.org/info/rfc5890>>.
- [RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <<https://www.rfc-editor.org/info/rfc6874>>.

## 11.2. Informative References

- [IEEE1588] IEEE, "IEEE Std 1588-2019 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control", IEEE Standard 1588, 2019, <<https://standards.ieee.org/content/ieee-standards/en/standard/1588-2019.html>>.

## Authors' Addresses

Heiko Gerstung  
Meinberg Funkuhren GmbH&Co.KG  
Lange Wand 9  
31812 Bad Pyrmont  
Germany

Phone: +49 5281 9309 0  
Email: [heiko.gerstung@meinberg.de](mailto:heiko.gerstung@meinberg.de)  
URI: <http://www.meinbergglobal.com/>

Marius Rohde  
Meinberg Funkuhren GmbH&Co.KG  
Lange Wand 9

31812 Bad Pyrmont  
Germany

Phone: +49 5281 9309 0  
Email: [marius.rohde@meinberg.de](mailto:marius.rohde@meinberg.de)  
URI: <http://www.meinbergglobal.com/>

Douglas Arnold  
Meinberg USA Inc.  
100 Stony Point Road Suite 110  
Santa Rosa, CA 95401  
United States of America

Phone: +1 877-PTP-1588  
Email: [doug.arnold@meinberg-usa.com](mailto:doug.arnold@meinberg-usa.com)  
URI: <http://www.meinbergglobal.com/>

Network Time Protocol  
Internet-Draft  
Intended status: Informational  
Expires: 17 May 2022

J. Guessing  
Nederlandse Publieke Omroep  
13 November 2021

NTPv5 use cases and requirements  
draft-guessing-ntp-ntp5-requirements-04

## Abstract

This document describes the use cases, requirements, and considerations that should be factored in the design of a successor protocol to supersede version 4 of the NTP protocol [RFC5905] presently referred to as NTP version 5 ("NTPv5"). This document is non-exhaustive and does not in its current version represent working group consensus.

## Note to Readers

\_RFC Editor: please remove this section before publication\_

Source code and issues for this draft can be found at <https://github.com/fiestajetsam/draft-guessing-ntp-ntp5-requirements> (<https://github.com/fiestajetsam/draft-guessing-ntp-ntp5-requirements>).

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 17 May 2022.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction	2
1.1. Notational Conventions	3
2. Use cases and existing deployments of NTP	3
3. Requirements	3
3.1. Resource management	3
3.2. Algorithms	4
3.3. Timescales	4
3.4. Leap seconds	5
3.5. Backwards compatibility to NTS and NTPv4	5
3.5.1. Dependent Specifications	5
3.6. Extensibility	5
3.7. Security	6
4. Non-requirements	6
4.1. Server malfeasance detection	6
5. Threat model	6
5.1. Delay-based attacks	6
5.2. Payload manipulation	7
5.3. Denial of Service and Amplification	7
6. IANA Considerations	7
7. Security Considerations	7
8. References	7
8.1. Normative References	7
8.2. Informative References	8
Appendix A. Acknowledgements	8
Author's Address	8

## 1. Introduction

NTP version 4 [RFC5905] has seen active use for over a decade, and within this time period the protocol has not only been extended to support new requirements but also fallen victim to vulnerabilities that have made it used for distributed denial of service (DDoS) amplification attacks.

### 1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 2. Use cases and existing deployments of NTP

There are several common scenarios for existing NTPv4 deployments; publicly accessible NTP services such as the NTP Pool [ntppool] are used to offer clock synchronisation for end users and embedded devices, ISP provided servers to synchronise devices such as customer-premises equipment where reduced accuracy may be tolerable. Depending on the network and path these deployments may be affected by variable latency as well as throttling or blocking by providers.

Data centres and cloud computing providers also have deployed and offer NTP services both for internal use and for customers, particularly where the network is unable to offer or does not require PTP [IEEE-1588-2008]. As these deployments are less likely to be constrained by network latency or power the potential for higher levels of accuracy and precision within the bounds of the protocol are possible.

## 3. Requirements

At a high level, NTPv5 should be a protocol that is capable of operating in both local networks and also over public internet connections where packet loss, delay, and even filtering may occur. It should be able to provide enough information for both basic time information as well as synchronisation.

### 3.1. Resource management

Historically there have been many documented instances of NTP servers taking a large increase in unauthorised traffic [ntp-misuse] and the design of NTPv5 must ensure the risk of these can be minimised to the fullest extent.

Servers SHOULD have a new identifier that peers use as reference, this SHOULD NOT be a FQDN, an IP address or identifier tied to a public certificate. Servers SHOULD be able to migrate and change their identifiers as stratum topologies or network configuration changes occur.

The protocol **MUST** have the capability for servers to notify clients that the service is unavailable, and clients **MUST** have clearly defined behaviours honouring this signalling. In addition servers **SHOULD** be able to communicate to clients that they should reduce their query interval rate when the server is under high bandwidth or has reduced capacity.

Clients **SHOULD** re-establish connections with servers at an interval to prevent attempting to maintain connectivity to a dead host and give network operators the ability to move traffic away from hosts in a timely manner.

The protocol **SHOULD** have provisions for deployments where Network Address Translation occurs, and define behaviours when NAT rebinding occurs. This should also not compromise any DDoS mitigation(s) that the protocol may define.

### 3.2. Algorithms

The use of algorithms describing functions such as clock filtering, selection and clustering **SHOULD** have agility, allowing for implementations to develop and deploy new algorithms independantly. Signalling of algorithm use or preference **SHOULD NOT** be transmitted by servers.

The working group should consider creating a separate informational document to describe an algorithm to assist with implementation, and to consider adopting future documents which describe new algorithms as they are developed. Specifying client algorithms separately from the protocol allows will allow NTPv5 to meet the needs of applications with a variety of network properties and performance requirements.

### 3.3. Timescales

The protocol **SHOULD** adopt a linear, monotonic timescale as the basis for communicating time. The format should meet sufficient scale and precision with resolution either meeting or exceeding NTPv4, and have a rollover date sufficiently far enough into the future that the protocol's complete obsolescence is most likely to occur first.

The timescale in addition to any other time sensitive information **MUST** be sufficient to calculate representations of both UTC and TAI. Through extensions the protocol **SHOULD** support additional timescale representations outside of the main specification, and all transmissions of time data **SHALL** indicate the timescale in use.

### 3.4. Leap seconds

Transmission of UTC leap second information MUST be included in the protocol in order for clients to generate a UTC representation but must be transmitted as separate information to the timescale. The specification SHOULD also be capable of transmitting upcoming leap seconds greater than 1 calendar day in advance.

Leap second smearing SHOULD NOT be applied to timestamps transmitted by the server, however this should not prevent implementers from applying leap second smearing between the client and any clock it is training.

### 3.5. Backwards compatibility to NTS and NTPv4

The support for compatibility with other protocols should not prevent addressing issues that have previously caused issues in deployments or cause ossification of the protocol.

Protocol ossification MUST be addressed to prevent existing NTPv4 deployments which incorrectly respond to clients posing as NTPv5 from causing issues. Forward prevention of ossification (for a potential NTPv6 protocol in the future) should also be taken into consideration.

The model for backward compatibility is servers that support multiple versions NTP and send a response in the same version as the request. This does not preclude servers from acting as a client in one version of NTP and a server in another.

#### 3.5.1. Dependent Specifications

Many other documents make use of NTP's data formats ([RFC5905] Section 6) for representing time, notably for media and packet timestamp measurements. Any changes to the data formats should consider the potential implementation complexity that may be incurred.

### 3.6. Extensibility

The protocol MUST have the capability to be extended, and that implementations MUST ignore unknown extensions. Unknown extensions received by a server from a lower stratum server SHALL not be added to response messages sent by the server receiving these extensions.

### 3.7. Security

Data authentication and optional data confidentiality **MUST** be integrated into the protocol, and downgrade attacks by an in-path attacker must be mitigated.

Cryptographic agility must be available, allowing for the protocol to update to the use of more secure cryptographic primitives as they are developed and as attacks and vulnerabilities with incumbent primitives are discovered. Intermediate devices such as hardware capable of performing timestamping of packets **SHOULD** be able to include information to packets in flight without requiring modification or removal of authentication or confidentiality on the packet.

Consideration must be given around how this will be incorporated into any applicable trust model. Downgrading attacks that could lead to an adversary disabling or removing encryption or authentication **MUST NOT** be possible in the design of the protocol.

## 4. Non-requirements

This section covers topics that are explicitly out of scope.

### 4.1. Server malfeasance detection

Detection and reporting of server malfeasance should remain out of scope as [I-D.ietf-ntp-rough-time] already provides this capability as a core functionality of the protocol.

## 5. Threat model

The assumptions that apply to all of the threats and risks within this section are based on observations of the use cases defined earlier in this document, and focus on external threats outside of the trust boundaries which may be in place within a network. Internal threats and risks such as a trusted operator are out of scope.

### 5.1. Delay-based attacks

The risk that an on-path attacker can delay packets between a client and server exists in all time protocols operating on insecure networks and its mitigations are limited within the protocol with a clock which is not yet synchronised. Increased path diversity and protocol support for synchronisation across multiple heterogeneous sources are likely the most effective mitigations.

## 5.2. Payload manipulation

Conversely on-path attackers who can manipulate timestamps could also speed up a client's clock, also resulting into drift-related malfunctions and errors such as incorrect expiration of public certificates on the affected hosts. An attacker may also manipulate other data in flight to disrupt service and cause de-synchronisation. In both cases having message authentication with a regular key rotation interval should mitigate; however consideration should be made for hardware based timestamping.

## 5.3. Denial of Service and Amplification

NTPv4 has previously suffered from DDoS amplification attacks using a combination of IP address spoofing with a private mode commands used in many NTP implementations, leading to an attacker being able to orders of magnitude of traffic to a victim IP address. Current mitigation uses a combination of disabling the use of private mode commands, in addition to encouraging network operators to implement BCP 38 [RFC2827]. Additional mitigations in future protocol specification should reduce the amplification factor in request/response payload sizes [drdos-amplification] through the use of padding and consideration of payload data.

## 6. IANA Considerations

This document makes no requests of IANA.

## 7. Security Considerations

As this document is intended to create discussion and consensus, it introduces no security considerations of its own.

## 8. References

### 8.1. Normative References

- [I-D.ietf-ntp-roughitime] Malhotra, A., Langley, A., Ladd, W., and M. Dansarie, "Roughitime", Work in Progress, Internet-Draft, draft-ietf-ntp-roughitime-05, 24 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-ntp-roughitime-05>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.

- [RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827, May 2000, <<https://www.rfc-editor.org/rfc/rfc2827>>.
- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/rfc/rfc5905>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.

## 8.2. Informative References

- [drdos-amplification] "Amplification and DRDoS Attack Defense -- A Survey and New Perspectives", n.d., <<https://arxiv.org/abs/1505.07892>>.
- [IEEE-1588-2008] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", n.d..
- [ntp-misuse] "NTP server misuse and abuse", n.d., <[https://en.wikipedia.org/wiki/NTP\\_server\\_misuse\\_and\\_abuse](https://en.wikipedia.org/wiki/NTP_server_misuse_and_abuse)>.
- [ntppool] "pool.ntp.org: the internet cluster of ntp servers", n.d., <<https://www.ntppool.org>>.

## Appendix A. Acknowledgements

The author would like to thank Doug Arnold and Hal Murray for contributions to this document, and would like to acknowledge Daniel Franke, Watson Ladd, Miroslav Lichvar for their existing documents and ideas. The author would also like to thank Angelo Moriondo, Franz Karl Achard, and Malcom McLean for providing the author with motivation.

## Author's Address

James Gruessing  
Nederlandse Publieke Omroep  
Postbus 26444

1202 JJ Hilversum  
Netherlands

Email: [james.ietf@gmail.com](mailto:james.ietf@gmail.com)

Internet Engineering Task Force  
Internet-Draft  
Updates: 5905 (if approved)  
Intended status: Standards Track  
Expires: 21 April 2022

M. Lichvar  
Red Hat  
18 October 2021

Alternative NTP port  
draft-ietf-ntp-alternative-port-02

Abstract

This document updates RFC 5905 to specify an alternative port for the Network Time Protocol (NTP) which is restricted to NTP messages that do not allow traffic amplification.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 21 April 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Requirements Language . . . . .	3
2. Alternative port - update to RFC 5905 . . . . .	3
3. IANA Considerations . . . . .	5
4. Security Considerations . . . . .	5
5. Acknowledgements . . . . .	5
6. References . . . . .	5
6.1. Normative References . . . . .	5
6.2. Informative References . . . . .	6
Author's Address . . . . .	6

## 1. Introduction

There are several modes specified for NTP. NTP packets in versions 2, 3, and 4 have a 3-bit field for the mode. Modes 1 (active), 2 (passive), 3 (client), 4 (server), and 5 (broadcast) are used for synchronization of clocks. They are specified in RFC 5905 [RFC5905]. Modes 6 and 7 are used for other purposes, like monitoring and remote management of NTP servers and clients. The mode 6 is specified in Control Messages Protocol for Use with Network Time Protocol Version 4 [I-D.ietf-ntp-mode-6-cmds].

The first group of modes typically does not allow any traffic amplification, i.e. the response is not larger than the request. An exception is Autokey [RFC5906], which allows an NTP response to be longer than the request, e.g. packets containing the Certificate Message or Cookie Message extension field. Autokey is rarely used. If it is enabled on a publicly accessible server, the access needs to be tightly controlled to limit denial-of-service (DoS) attacks exploiting the amplification.

The modes 6 and 7 of NTP allow significant traffic amplification, which has been exploited in large-scale DoS attacks on the Internet. Publicly accessible servers that support these modes need to be configured to not respond to requests using the modes, as recommended in BCP 233 [RFC8633], but the number of servers that still do that is significant enough to require specific mitigations.

Network operators have implemented different mitigations. They are not documented and may change over time. Some of the mitigations that have been observed are:

1. Blocked UDP packets with destination or source port 123
2. Blocked UDP packets with destination or source port 123 and specific length (e.g. longer than 48 octets)

3. Blocked UDP packets with destination or source port 123 and NTP mode 6 or 7
4. Limited rate of UDP packets with destination or source port 123

From those, only the 3rd approach does not have an impact on synchronization of clocks with NTP. However, this mitigation can be implemented only on devices which can inspect the UDP payload.

The number of public servers in the pool.ntp.org project has dropped since 2013, when the large-scale attacks started.

The length-specific filtering and rate limiting has an impact on the Network Time Security [RFC8915] authentication, which uses extension fields in NTPv4 packets.

This document specifies an alternative port for NTP which is restricted to a subset of the NTP protocol which does not allow amplification in order to enable safe synchronization of clocks in networks where the port 123 is blocked or rate limited.

### 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 2. Alternative port - update to RFC 5905

The table in "Figure 6: Global Parameters" in Section 7.2 of [RFC5905] is extended with:

Name	Value	Description
ALTPORT	TBD	Alternative NTP port

Table 1

The following text from Section 9.1 of [RFC5905]:

srcport: UDP port number of the server or reference clock. This becomes the destination port number in packets sent from this association. When operating in symmetric modes (1 and 2), this field must contain the NTP port number PORT (123) assigned by the IANA. In other modes, it can contain any number consistent with local policy.

is replaced with:

srcport: UDP port number of the server or reference clock. This becomes the destination port number in packets sent from this association. When operating in symmetric modes (1 and 2), this field must contain the NTP port number PORT (123) or the alternative NTP port ALTPORT (TBD) assigned by the IANA. In other modes, it can contain any number consistent with local policy.

The following text is added to the Section 9.1:

The port ALTPORT (TBD) is an alternative port to the port PORT (123). The protocol and the format of NTP packets sent from and to this port is unchanged. Both NTP requests and responses MAY be sent from the alternative port. An NTP packet MUST NOT be sent from the alternative port if it is a response which has a longer UDP payload than the request, or the number of NTP packets in a single response is larger than one.

Only modes 1 (active), 2 (passive), 3 (client), 4 (server), and 5 (broadcast) are generally usable on this port.

An NTP server that supports the alternative port MUST receive requests in the client mode on both the PORT (123) and ALTPORT (TBD) ports. If it responds, it MUST send the response from the port which received the request. If the server supports an NTP extension field, it MUST verify for each response that it is not longer than the request.

When an NTP client is started, it SHOULD send the first request to the alternative port. The client SHOULD alternate between the two ports until a valid response is received. The client MAY send a limited number of requests to both ports at the same time in order to speed up the discovery of the responding port. When both ports are responding, the client SHOULD prefer the alternative port.

An NTP server which supports NTS SHOULD include the NTPv4 Port Negotiation record in NTS-KE responses to specify the alternative port as the port to which the client should send NTP requests.

In the symmetric modes (active and passive) NTP packets are considered to be requests and responses at the same time. Therefore, two peers using the alternative port MUST send packets with an equal length in order to synchronize with each other. The peers MAY still use different polling intervals as packets sent at subsequent polls are considered to be separate requests and responses.

### 3. IANA Considerations

IANA is requested to allocate the following port in the Service Name and Transport Protocol Port Number Registry [RFC6335]:

Service Name: ntp-alt

Transport Protocol: udp

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: Network Time Protocol

Reference: [[this memo]]

Port Number: [[TBD]], selected by IANA from the System Port range

### 4. Security Considerations

A Man-in-the-middle (MITM) attacker can selectively block requests sent to the alternative port to force a client to select the original port and get a degraded NTP service with a significant packet loss. The client needs to periodically try the alternative port to recover from the degraded service when the attack stops.

### 5. Acknowledgements

The author would like to thank Daniel Franke, Dhruv Dhody, Ragnar Sundblad, and Steven Sommars for their useful comments.

### 6. References

#### 6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

## 6.2. Informative References

- [I-D.ietf-ntp-mode-6-cmds] Haberman, B., "Control Messages Protocol for Use with Network Time Protocol Version 4", Work in Progress, Internet-Draft, draft-ietf-ntp-mode-6-cmds-10, 28 September 2020, <<https://www.ietf.org/archive/id/draft-ietf-ntp-mode-6-cmds-10.txt>>.
- [RFC5906] Haberman, B., Ed. and D. Mills, "Network Time Protocol Version 4: Autokey Specification", RFC 5906, DOI 10.17487/RFC5906, June 2010, <<https://www.rfc-editor.org/info/rfc5906>>.
- [RFC8633] Reilly, D., Stenn, H., and D. Sibold, "Network Time Protocol Best Current Practices", BCP 223, RFC 8633, DOI 10.17487/RFC8633, July 2019, <<https://www.rfc-editor.org/info/rfc8633>>.
- [RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <<https://www.rfc-editor.org/info/rfc8915>>.

## Author's Address

Miroslav Lichvar  
Red Hat  
Purkynova 115  
612 00 Brno  
Czech Republic

Email: [mlichvar@redhat.com](mailto:mlichvar@redhat.com)

Internet Engineering Task Force  
Internet-Draft  
Updates: 5905 (if approved)  
Intended status: Standards Track  
Expires: 21 April 2022

M. Lichvar  
Red Hat  
A. Malhotra  
Boston University  
18 October 2021

NTP Interleaved Modes  
draft-ietf-ntp-interleaved-modes-07

Abstract

This document extends the specification of Network Time Protocol (NTP) version 4 in RFC 5905 with special modes called the NTP interleaved modes, that enable NTP servers to provide their clients and peers with more accurate transmit timestamps that are available only after transmitting NTP packets. More specifically, this document describes three modes: interleaved client/server, interleaved symmetric, and interleaved broadcast.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 21 April 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Requirements Language . . . . .	4
2. Interleaved Client/server mode . . . . .	4
3. Interleaved Symmetric mode . . . . .	9
4. Interleaved Broadcast mode . . . . .	10
5. Protocol Failures . . . . .	11
6. Security Considerations . . . . .	13
7. IANA Considerations . . . . .	14
8. Acknowledgements . . . . .	14
9. References . . . . .	14
9.1. Normative References . . . . .	14
9.2. Informative References . . . . .	15
Authors' Addresses . . . . .	15

## 1. Introduction

RFC 5905 [RFC5905] describes the operations of NTPv4 in a client/server, symmetric, and broadcast mode. The transmit and receive timestamps are two of the four timestamps included in every NTPv4 packet used for time synchronization.

For a highly accurate and stable synchronization, the transmit and receive timestamp should be captured close to the beginning of the actual transmission and the end of the reception respectively. An asymmetry in the timestamping causes the offset measured by NTP to have an error.

There are at least four options where a timestamp of an NTP packet may be captured with a software NTP implementation running on a general-purpose operating system:

1. User space (software)
2. Network device driver or kernel (software)
3. Data link layer (hardware - MAC chip)

#### 4. Physical layer (hardware - PHY chip)

Software timestamps captured in user space in the NTP implementation itself are least accurate. They do not include system calls used for sending and receiving packets, processing and queuing delays in the system, network device drivers, and hardware. Hardware timestamps captured at the physical layer are most accurate.

A transmit timestamp captured in the driver or hardware is more accurate than the user-space timestamp, but it is available to the NTP implementation only after it sent the packet using a system call. The timestamp cannot be included in the packet itself unless the driver or hardware supports NTP and can modify the packet before or during the actual transmission.

The protocol described in RFC 5905 does not specify any mechanism for a server to provide its clients and peers with a more accurate transmit timestamp that is known only after the transmission. A packet that strictly follows RFC 5905, i.e. it contains a transmit timestamp corresponding to the packet itself, is said to be in basic mode.

Different mechanisms could be used to exchange timestamps known after the transmission. The server could respond to each request with two packets. The second packet would contain the transmit timestamp corresponding to the first packet. However, such a protocol would enable a traffic amplification attack, or it would use packets with an asymmetric length, which would cause an asymmetry in the network delay and an error in the measured offset.

This document describes an interleaved client/server, interleaved symmetric, and interleaved broadcast mode. In these modes, the server sends a packet which contains a transmit timestamp corresponding to the transmission of the previous packet that was sent to the client or peer. This transmit timestamp can be captured in any software or hardware component involved in the transmission of the packet. Both servers and clients/peers are required to keep some state specific to the interleaved mode.

An NTPv4 implementation that supports the client/server and broadcast interleaved modes interoperates with NTPv4 implementations without this capability. A peer using the symmetric interleaved mode does not fully interoperate with a peer which does not support it. The mode needs to be configured specifically for each symmetric association.

The interleaved modes do not change the NTP packet header format and do not use new extension fields. The negotiation is implicit. The protocol is extended with new values that can be assigned to the origin and transmit timestamp. Servers and peers check the origin timestamp to detect requests conforming to the interleaved mode. A response can be valid only in one mode. If a client or peer that does not support interleaved mode received a response conforming to the interleaved mode, it would be rejected as bogus.

An explicit negotiation would require a new extension field. RFC 5905 does not specify how servers should handle requests with an unknown extension field. The original use of extension fields was authentication with Autokey [RFC5906], which cannot be negotiated. Some existing implementations do not respond to requests with unknown extension fields. This behavior would prevent clients from reliably detecting support for the interleaved mode.

Requests and responses cannot always be formed in interleaved mode. It cannot be used exclusively. Servers, clients, and peers that support the interleaved mode need to support also the basic mode.

This document assumes familiarity with RFC 5905.

### 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 2. Interleaved Client/server mode

The interleaved client/server mode is similar to the basic client/server mode. The difference between the two modes is in the values saved to the origin and transmit timestamp fields.

The origin timestamp is a cookie which is used to detect that a received packet is a response to the last packet sent in the other direction of the association. It is a copy of one of the timestamps from the packet to which it is responding, or zero if it is not a response. Servers following RFC 5905 ignore the origin timestamp in client requests. A server response which does not have a matching origin timestamp is called bogus.

A client request in the basic mode has an origin timestamp equal to the transmit timestamp from the last valid server response, or is zero (which indicates the first request of the association). A

server response in the basic mode has an origin timestamp equal to the transmit timestamp from the client request. The transmit timestamp in the response corresponds to the transmission of the response in which the timestamp is contained.

A client request in the interleaved mode has an origin timestamp equal to the receive timestamp from the last valid server response. A server response in the interleaved mode has an origin timestamp equal to the receive timestamp from the client request. The transmit timestamp in the response corresponds to the transmission of the previous response which had the receive timestamp equal to the origin timestamp from the request.

A server which supports the interleaved mode needs to save pairs of local receive and transmit timestamps. The server **SHOULD** discard old timestamps to limit the amount of memory needed to support clients using the interleaved mode. The server **MAY** separate the timestamps by IP addresses, but it **SHOULD NOT** separate them by port numbers to support clients that change their port between requests, as recommended in RFC 9109 [RFC9109].

The server **MAY** restrict the interleaved mode to specific IP addresses and/or authenticated clients.

Both servers and clients that support the interleaved mode **MUST NOT** send a packet that has a transmit timestamp equal to the receive timestamp in order to reliably detect whether received packets conform to the interleaved mode. One way to ensure that is to increment the transmit timestamp by 1 unit (i.e. about 1/4 of a nanosecond) if the two timestamps are equal, or a new timestamp can be generated.

The transmit and receive timestamps in server responses need to be unique to prevent two different clients from sending requests with the same origin timestamp and the server responding in the interleaved mode with an incorrect transmit timestamp. If the timestamps are not guaranteed to be monotonically increasing, the server **SHOULD** check that the transmit and receive timestamps are not already saved as a receive timestamp of a previous request (from the same IP address if the server separates timestamps by addresses), and generate a new timestamp if necessary.

When the server receives a request from a client, it **SHOULD** respond in the interleaved mode if the following conditions are met:

1. The request does not have a receive timestamp equal to the transmit timestamp.

2. The origin timestamp from the request matches the local receive timestamp of a previous request that the server has saved (for the IP address if it separates timestamps by addresses).

A response in the interleaved mode MUST contain the transmit timestamp of the response which contained the receive timestamp matching the origin timestamp from the request. The server SHOULD drop the timestamps after sending the response. The receive timestamp MUST NOT be used again to detect a request conforming to the interleaved mode.

If the conditions are not met (i.e. the request is not detected to conform to the interleaved mode), the server MUST NOT respond in the interleaved mode. The server MAY always respond in the basic mode. In any case, the server SHOULD save the new receive and transmit timestamps.

The first request from a client is always in the basic mode and so is the server response. It has a zero origin timestamp and zero receive timestamp. Only when the client receives a valid response from the server, it will be able to send a request in the interleaved mode.

The protocol recovers from packet loss. When a client request or server response is lost, the client will use the same origin timestamp in the next request. The server can respond in the interleaved mode if it still has the timestamps corresponding to the origin timestamp. If the server already responded to the timestamp in the interleaved mode, or it had to drop the timestamps for other reasons, it will respond in the basic mode and save new timestamps, which will enable an interleaved response to the subsequent request. The client SHOULD limit the number of requests in the interleaved mode between server responses to prevent processing of very old timestamps in case a large number of consecutive requests is lost.

An example of packets in a client/server exchange using the interleaved mode is shown in Figure 1. The packets in the basic and interleaved mode are indicated with B and I respectively. The timestamps  $t1^*$ ,  $t3^*$  and  $t11^*$  point to the same transmissions as  $t1$ ,  $t3$  and  $t11$ , but they may be less accurate. The first exchange is in the basic mode followed by a second exchange in the interleaved mode. For the third exchange, the client request is in the interleaved mode, but the server response is in the basic mode, because the server did not have the pair of timestamps  $t6$  and  $t7$  (e.g. they were dropped to save timestamps for other clients using the interleaved mode).

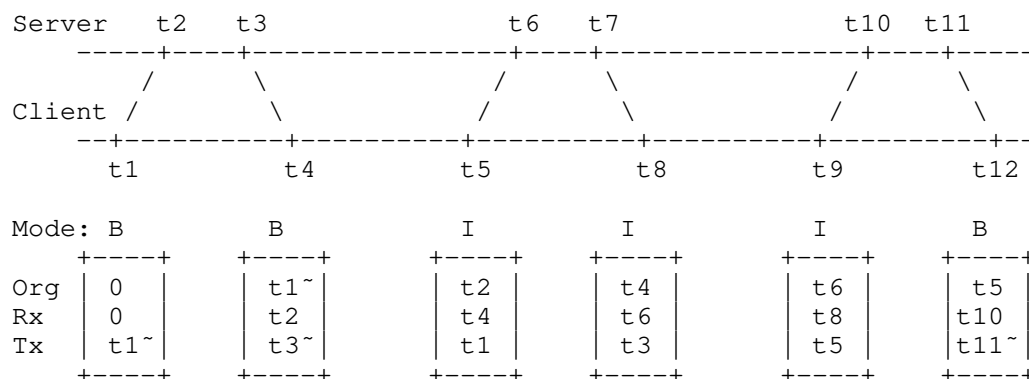


Figure 1: Packet timestamps in interleaved client/server mode

When the client receives a response from the server, it performs the tests described in RFC 5905. Two of the tests are modified for the interleaved mode:

1. The check for duplicate packets SHOULD compare both receive and transmit timestamps in order to not drop a valid response in the interleaved mode if it follows a response in the basic mode and they contain the same transmit timestamp.
2. The check for bogus packets SHOULD compare the origin timestamp with both transmit and receive timestamps from the request. If the origin timestamp is equal to the transmit timestamp, the response is in the basic mode. If the origin timestamp is equal to the receive timestamp, the response is in the interleaved mode.

The client SHOULD NOT update its NTP state when an invalid response is received, to not lose the timestamps which will be needed to complete a measurement when the subsequent response in the interleaved mode is received.

If the packet passed the tests and conforms to the interleaved mode, the client can compute the offset and delay using the formulas from RFC 5905 and one of two different sets of timestamps. The first set is RECOMMENDED for clients that filter measurements based on the delay. The corresponding timestamps from Figure 1 are written in parentheses.

T1 - local transmit timestamp of the previous request (t1)

T2 - remote receive timestamp from the previous response (t2)

T3 - remote transmit timestamp from the latest response (t3)

T4 - local receive timestamp of the previous response (t4)

The second set gives a more accurate measurement of the current offset, but the delay is much more sensitive to a frequency error between the server and client due to a much longer interval between T1 and T4.

T1 - local transmit timestamp of the latest request (t5)

T2 - remote receive timestamp from the latest response (t6)

T3 - remote transmit timestamp from the latest response (t3)

T4 - local receive timestamp of the previous response (t4)

Clients MAY filter measurements based on the mode. The maximum number of dropped measurements in the basic mode SHOULD be limited in case the server does not support or is not able to respond in the interleaved mode. Clients that filter measurements based on the delay will implicitly prefer measurements in the interleaved mode over the basic mode, because they have a shorter delay due to a more accurate transmit timestamp (T3).

The server MAY limit saving of the receive and transmit timestamps to requests which have an origin timestamp specific to the interleaved mode in order to not waste resources on clients using the basic mode. Such an optimization will delay the first interleaved response of the server to a client by one exchange.

A check for a non-zero origin timestamp works with SNTP clients that always set the timestamp to zero and clients that implement NTP data minimization [I-D.ietf-ntp-data-minimization]. From the server's point of view, such clients start a new association with each request.

To avoid searching the saved receive timestamps for non-zero origin timestamps from requests conforming to the basic mode, the server can encode in low-order bits of the receive and transmit timestamps below precision of the clock a flag indicating whether the timestamp is a receive timestamp. If the server receives a request with a non-zero origin timestamp which does not indicate it is a receive timestamp of the server, the request does not conform to the interleaved mode and it is not necessary to perform the search and/or save the new receive and transmit timestamp.

### 3. Interleaved Symmetric mode

The interleaved symmetric mode uses the same principles as the interleaved client/server mode. A packet in the interleaved symmetric mode has a transmit timestamp which corresponds to the transmission of the previous packet sent to the peer and an origin timestamp equal to the receive timestamp from the last packet received from the peer.

To enable synchronization in both directions of a symmetric association, both peers need to support the interleaved mode. For this reason, it SHOULD be disabled by default and enabled with an option in the configuration of the active side of the association.

In order to prevent the peer from matching the transmit timestamp with an incorrect packet when the peers' transmissions do not alternate (e.g. they use different polling intervals) and a previous packet was lost, the use of the interleaved mode in symmetric associations requires additional restrictions.

Peers which have an association need to count valid packets received between their transmissions to determine in which mode a packet should be formed. A valid packet in this context is a packet which passed all NTP tests for duplicate, replayed, bogus, and unauthenticated packets. Other received packets may update the NTP state to allow the (re)initialization of the association, but they do not change the selection of the mode.

A peer A SHOULD send a peer B a packet in the interleaved mode only when all of the following conditions are met:

1. The peer A has an active association with the peer B which was specified with the option enabling the interleaved mode, OR the peer A received at least one valid packet in the interleaved mode from the peer B.
2. The peer A did not send a packet to the peer B since it received the last valid packet from the peer B.
3. The previous packet that the peer A sent to the peer B was the only response to a packet received from the peer B.

The first condition is needed for compatibility with implementations that do not support or are not configured for the interleaved mode. The other conditions prevent a missing response from causing a mismatch between the remote transmit (T2) and local receive timestamp (T3), which would cause a large error in the measured offset and delay.

An example of packets exchanged in a symmetric association is shown in Figure 2. The minimum polling interval of the peer A is twice as long as the maximum polling interval of the peer B. The first packets sent by the peers are in the basic mode. The second and third packet sent by the peer A is in the interleaved mode. The second packet sent by the peer B is in the interleaved mode, but the following packets sent by the peer B are in the basic mode, because multiple responses are sent per request.

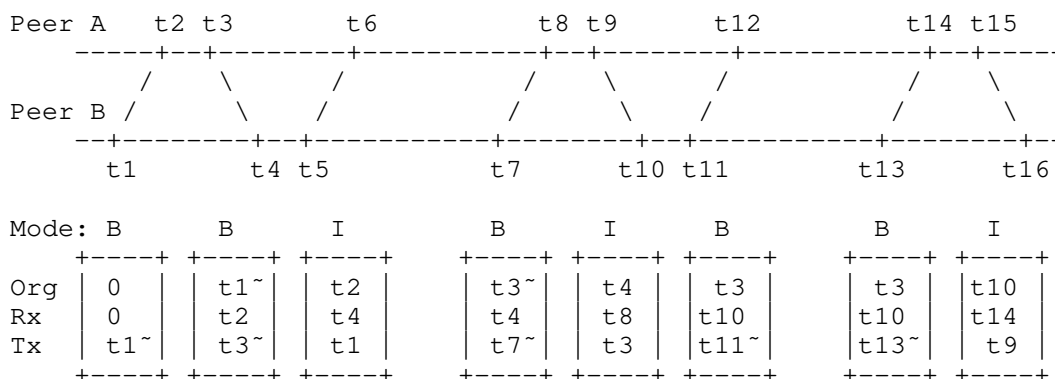


Figure 2: Packet timestamps in interleaved symmetric mode

If the peer A has no association with the peer B and it responds with symmetric passive packets, it does not need to count the packets in order to meet the restrictions, because each request has at most one response. The peer SHOULD process the requests in the same way as a server which supports the interleaved client/server mode. It MUST NOT respond in the interleaved mode if the request was not in the interleaved mode.

The peers SHOULD compute the offset and delay using one of the two sets of timestamps specified in the client/server section. They MAY switch between them to minimize the interval between T1 and T4 in order to reduce the error in the measured delay.

#### 4. Interleaved Broadcast mode

A packet in the interleaved broadcast mode contains two transmit timestamps. One corresponds to the packet itself and is saved in the transmit timestamp field. The other corresponds to the previous packet and is saved in the origin timestamp field. The packet is compatible with the basic mode, which uses a zero origin timestamp.

An example of packets sent in the broadcast mode is shown in Figure 3.

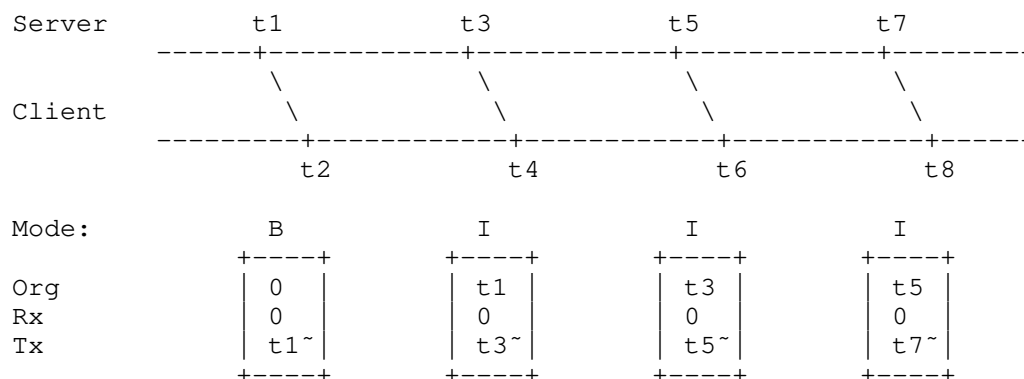


Figure 3: Packet timestamps in interleaved broadcast mode

A client which does not support the interleaved mode ignores the origin timestamp and processes all packets as if they were in the basic mode.

A client which supports the interleaved mode SHOULD check if the origin timestamp is not zero to detect packets in the interleaved mode. The client SHOULD also compare the origin timestamp with the transmit timestamp from the previous packet to detect lost packets. If the difference is larger than a specified maximum (e.g. 1 second), the packet SHOULD NOT be used for synchronization in the interleaved mode.

The client SHOULD compute the offset using the origin timestamp from the received packet and the local receive timestamp of the previous packet. If the client needs to measure the network delay, it SHOULD use the interleaved client/server mode.

## 5. Protocol Failures

An incorrect client implementation of the basic mode (RFC 5905) can work reliably with servers that implement only the basic mode, but the protocol can fail intermittently with servers that implement the interleaved mode.

If the client sets the origin timestamp to other values than the transmit timestamp from the last valid server response, or zero, the origin timestamp can match a receive timestamp of a previous server response (possibly to a different client), causing an unexpected interleaved response. The client is expected to drop the response as bogus. If it did not check for bogus packets, it would be vulnerable to off-path attacks.

If the client set the origin timestamp to a constant non-zero value, this mismatch would be expected to happen once per the NTP era (about 136 years) if the NTP server was responding at the maximum rate needed to go through all timestamp values (about 2 billion responses per second). With lower rates of requests the chance of hitting a server timestamp decreases proportionally.

The worst case of this failure would be a client that specifically sets the origin timestamp to the server's receive timestamp, i.e. the client accidentally implemented the interleaved mode, but it does not accept interleaved responses. This client would still be able to synchronize its clock. It would drop interleaved responses as bogus and set the origin timestamp to the receive timestamp from the last valid response in the basic mode. As servers are required to not respond twice to the same origin timestamp in the interleaved mode, at least every other response would be in the basic mode and accepted by the client.

Intermittent protocol failures can be caused also by an incorrect server implementation of the interleaved mode. A server which does not ensure the receive and transmit timestamps in its responses are unique in a sufficiently long interval can misinterpret requests formed correctly in the basic mode as interleaved and respond in the interleaved mode. The response would be dropped by the client as bogus.

A duplicated server receive timestamp can cause an expected interleaved response to contain a transmit timestamp which does not correspond to the transmission of the previous response from which the client copied the receive timestamp to the origin timestamp in the request, but a different response which contained the same receive timestamp. The response would be accepted by the client with a small error in the transmit timestamp equal to the difference between the transmit timestamps of the two different responses. As the two requests to which the responses responded were received at the same time (according to the server's clock), the two transmissions would be expected to be close to each other and the difference between them would be comparable to the error a basic response normally has in its transmit timestamp.

One reason for a duplicated server timestamp can be a large backward step of the server's clock. If the timestamps the server has saved do not fully cover the second pass of the clock over the repeated interval, two requests received in different passes of the clock can get the same receive timestamp. The client which made the first request can get the transmit timestamp corresponding to the transmission of the second response. From the server's point of view, the error of the transmit timestamp would be still small, but

from the client's point of view the server already failed when it made the step as it was serving wrong time before or after the step with a much larger error than the error caused by the protocol failure.

## 6. Security Considerations

The security considerations of time protocols in general are discussed in RFC 7384 [RFC7384], and specifically the security considerations of NTP are discussed in RFC 5905.

Security issues that apply to the basic modes apply also to the interleaved modes. They are described in The Security of NTP's Datagram Protocol [SECNTP].

Clients and peers SHOULD NOT leak the receive timestamp in packets sent to other peers or clients (e.g. as a reference timestamp) to prevent off-path attackers from easily getting the origin timestamp needed to make a valid response in the interleaved mode.

Clients using the interleaved mode SHOULD randomize all bits of both receive and transmit timestamps, as recommended for the transmit timestamp in the NTP client data minimization [I-D.ietf-ntp-data-minimization], to make it more difficult for off-path attackers to guess the origin timestamp in the server response.

The client data minimization cannot be fully implemented in the interleaved mode. The origin timestamp cannot be zeroed out, which makes the clients more vulnerable to tracking as they move between networks.

Attackers can force the server to drop its timestamps in order to prevent clients from getting an interleaved response. They can send a large number of requests, send requests with a spoofed source address, or replay an authenticated request if the interleaved mode is enabled only for authenticated clients. Clients SHOULD NOT rely on servers to be able to respond in the interleaved mode.

Protecting symmetric associations in the interleaved mode against replay attacks is even more difficult than in the basic mode. In both modes, the NTP state needs to be protected between the reception of the last non-replayed response and transmission of the next request in order for the request to contain the origin timestamp expected by the peer. The difference is in the timestamps needed to complete a measurement. In the basic mode only one valid response is needed at a time and it is used as soon as it is received, but the interleaved mode needs two consecutive valid responses. The NTP state needs to be protected all the time to not lose the timestamps which are needed to complete the measurement when the second response is received.

## 7. IANA Considerations

This memo includes no request to IANA.

## 8. Acknowledgements

The interleaved modes described in this document are based on the implementation written by David Mills in the NTP project (<http://www.ntp.org>). The specification of the broadcast mode is based purely on this implementation. The specification of the symmetric mode has some modifications. The client/server mode is specified as a new mode compatible with the symmetric mode, similarly to the basic symmetric and client/server modes.

The authors would like to thank Theresa Enghardt, Daniel Franke, Benjamin Kaduk, Erik Kline, Tal Mizrahi, Steven Sommars, Harlan Stenn, and Kristof Teichel for their useful comments.

## 9. References

### 9.1. Normative References

- [I-D.ietf-ntp-data-minimization]  
Franke, D. F. and A. Malhotra, "NTP Client Data Minimization", Work in Progress, Internet-Draft, draft-ietf-ntp-data-minimization-04, 25 March 2019, <<https://www.ietf.org/archive/id/draft-ietf-ntp-data-minimization-04.txt>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

## 9.2. Informative References

- [RFC5906] Haberman, B., Ed. and D. Mills, "Network Time Protocol Version 4: Autokey Specification", RFC 5906, DOI 10.17487/RFC5906, June 2010, <<https://www.rfc-editor.org/info/rfc5906>>.
- [RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <<https://www.rfc-editor.org/info/rfc7384>>.
- [RFC9109] Gont, F., Gont, G., and M. Lichvar, "Network Time Protocol Version 4: Port Randomization", RFC 9109, DOI 10.17487/RFC9109, August 2021, <<https://www.rfc-editor.org/info/rfc9109>>.
- [SECNTP] Malhotra, A., Gundy, M. V., Varia, M., Kennedy, H., Gardner, J., and S. Goldberg, "The Security of NTP's Datagram Protocol", 2016, <<http://eprint.iacr.org/2016/1006>>.

## Authors' Addresses

Miroslav Lichvar  
Red Hat  
Purkynova 115  
612 00 Brno  
Czech Republic

Email: [mlichvar@redhat.com](mailto:mlichvar@redhat.com)

Aanchal Malhotra  
Boston University  
111 Cummington St  
Boston, 02215  
United States of America

Email: [aanchal4@bu.edu](mailto:aanchal4@bu.edu)

Internet Engineering Task Force  
Internet-Draft  
Intended status: Informational  
Expires: 21 November 2021

A. Malhotra  
Boston University  
A. Langley  
Google  
W. Ladd  
Cloudflare  
M. Dansarie  
20 May 2021

Roughtime  
draft-ietf-ntp-roughtime-05

## Abstract

This document specifies Roughtime - a protocol that aims to achieve rough time synchronization while detecting servers that provide inaccurate time and providing cryptographic proof of their malfeasance.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 21 November 2021.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Requirements Language . . . . .	4
3. Protocol Overview . . . . .	4
4. The Guarantee . . . . .	5
5. Message Format . . . . .	5
5.1. Data Types . . . . .	6
5.1.1. int32 . . . . .	6
5.1.2. uint32 . . . . .	6
5.1.3. uint64 . . . . .	6
5.1.4. Tag . . . . .	7
5.1.5. Timestamp . . . . .	7
5.2. Header . . . . .	7
6. Protocol Details . . . . .	8
6.1. Requests . . . . .	9
6.1.1. VER . . . . .	9
6.1.2. NONC . . . . .	9
6.2. Responses . . . . .	9
6.2.1. SIG . . . . .	9
6.2.2. VER . . . . .	10
6.2.3. NONC . . . . .	10
6.2.4. PATH . . . . .	10
6.2.5. SREP . . . . .	10
6.2.6. CERT . . . . .	11
6.2.7. INDX . . . . .	11
6.3. The Merkle Tree . . . . .	12
6.3.1. Root Value Validity Check Algorithm . . . . .	12
6.4. Validity of Response . . . . .	13
7. Integration Into NTP . . . . .	13
8. Grease . . . . .	14
9. RoughTime Servers . . . . .	14
10. Acknowledgements . . . . .	14
11. IANA Considerations . . . . .	14
11.1. Service Name and Transport Protocol Port Number Registry . . . . .	14
11.2. RoughTime Version Registry . . . . .	15
11.3. RoughTime Tag Registry . . . . .	15
12. Security Considerations . . . . .	17

13. Privacy Considerations . . . . .	18
14. References . . . . .	18
14.1. Normative References . . . . .	18
14.2. Informative References . . . . .	19
Appendix A. Terms and Abbreviations . . . . .	20
Authors' Addresses . . . . .	21

## 1. Introduction

Time synchronization is essential to Internet security as many security protocols and other applications require synchronization [RFC7384] [MCBG]. Unfortunately widely deployed protocols such as the Network Time Protocol (NTP) [RFC5905] lack essential security features, and even newer protocols like Network Time Security (NTS) [RFC8915] lack mechanisms to ensure that the servers behave correctly. Authenticating time servers prevents network adversaries from modifying time packets, but an authenticated time server still has full control over the contents of the time packet and may transmit incorrect time. The RoughTime protocol provides cryptographic proof of malfeasance, enabling clients to detect and prove to a third party a server's attempts to influence the time a client computes.

Protocol	Authenticated Server	Server Malfeasance Evidence
NTP, Chronos	N	N
NTP-MAC	Y*	N
NTP-Autokey	Y**	N
NTS	Y	N
RoughTime	Y	Y

Table 1: Security Properties of current protocols.

Y\* For security issues with symmetric-key based NTP-MAC authentication, please refer to RFC 8573 [RFC8573].

Y\*\* For security issues with Autokey Public Key Authentication, refer to [Autokey].

- \* If a server's timestamps do not fit into the time context of other servers' responses, then a Roughtime client can cryptographically prove this misbehavior to third parties. This helps detect dishonest or malfunctioning servers.
- \* A Roughtime client can roughly detect (with no absolute guarantee) a delay attack [DelayAttacks] but can not cryptographically prove this to a third party. However such attacks expand the round trip time between request and response.
- \* Note that delay attacks cannot be detected/stopped by any protocol. Delay attacks can not, however, undermine the security guarantees provided by Roughtime.
- \* Although delay attacks cannot be prevented, they can be limited to a predetermined upper bound. This can be done by defining a maximal tolerable Round Trip Time (RTT) value, MAX-RTT, that a Roughtime client is willing to accept. A Roughtime client can measure the RTT of every request-response handshake and compare it to MAX-RTT. If the RTT exceeds MAX-RTT, the corresponding measurement is discarded. When this approach is used, the maximal time error that can be caused by a delay attack is MAX-RTT/2. It should be noted that this approach assumes that the nature of the system is known to the client, including reasonable upper bounds on the RTT value.

## 2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 3. Protocol Overview

Roughtime is a protocol for rough time synchronization that enables clients to provide cryptographic proof of server malfeasance. It does so by having responses from servers include a signature over a value derived from a nonce in the client request. This provides cryptographic proof that the timestamp was issued after the server received the client's request. The derived value included in the server's response is the root of a Merkle tree which includes the hash of the client's nonce as the value of one of its leaf nodes. This enables the server to amortize the relatively costly signing operation over a number of client requests.

Single server mode: At its most basic level, Roughtime is a one round protocol in which a completely fresh client requests the current time and the server sends a signed response. The response includes a timestamp and a radius used to indicate the server's certainty about the reported time. For example, a radius of 1,000,000 microseconds means the server is absolutely confident that the true time is within one second of the reported time.

The server proves freshness of its response as follows. The client's request contains a nonce which the server incorporates into its signed response. The client can verify the server's signatures and - provided that the nonce has sufficient entropy - this proves that the signed response could only have been generated after the nonce.

#### 4. The Guarantee

A Roughtime server guarantees that a response to a query sent at  $t_1$ , received at  $t_2$ , and with timestamp  $t_3$  has been created between the transmission of the query and its reception. If  $t_3$  is not within that interval, a server inconsistency may be detected and used to impeach the server. The propagation of such a guarantee and its use of type synchronization is discussed in Section 7. No delay attacker may affect this: they may only expand the interval between  $t_1$  and  $t_2$ , or of course stop the measurement in the first place.

#### 5. Message Format

Roughtime messages are maps consisting of one or more (tag, value) pairs. They start with a header, which contains the number of pairs, the tags, and value offsets. The header is followed by a message values section which contains the values associated with the tags in the header. Messages MUST be formatted according to Figure 1 as described in the following sections.

Messages MAY be recursive, i.e. the value of a tag can itself be a Roughtime message.

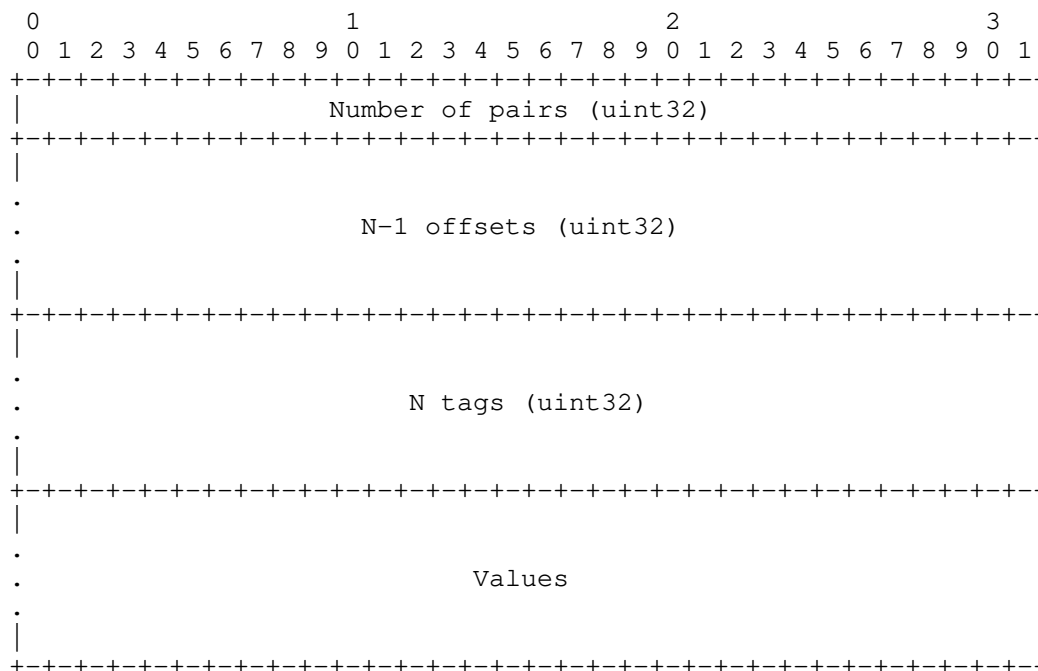


Figure 1: Roughtime Message Format

## 5.1. Data Types

### 5.1.1. int32

An int32 is a 32 bit signed integer. It is serialized least significant byte first in sign-magnitude representation with the sign bit in the most significant bit. The negative zero value (0x80000000) MUST NOT be used and any message with it is syntactically invalid and MUST be ignored.

### 5.1.2. uint32

A uint32 is a 32 bit unsigned integer. It is serialized with the least significant byte first.

### 5.1.3. uint64

A uint64 is a 64 bit unsigned integer. It is serialized with the least significant byte first.

#### 5.1.4. Tag

Tags are used to identify values in RoughTime messages. A tag is a uint32 but may also be listed in this document as a sequence of up to four ASCII characters [RFC0020]. ASCII strings shorter than four characters can be unambiguously converted to tags by padding them with zero bytes. For example, the ASCII string "NONC" would correspond to the tag 0x434e4f4e and "PAD" would correspond to 0x00444150. Note that when encoded into a message the ASCII values will be in the corresponding order.

#### 5.1.5. Timestamp

A timestamp is a uint64 interpreted in the following way. The most significant 3 bytes contain the integer part of a Modified Julian Date (MJD). The least significant 5 bytes is a count of the number of microseconds since midnight on that day.

The MJD is the number of UTC days since 17 November 1858 [ITU-R\_TF.457-2]. It is useful to note that 1 January 1970 is 40,587 days after 17 November 1858.

Note that, unlike NTP, this representation does not use the full number of bits in the fractional part and that days with leap seconds will have more or fewer than the nominal 86,400,000,000 microseconds.

#### 5.2. Header

All RoughTime messages start with a header. The first four bytes of the header is the uint32 number of tags  $N$ , and hence of (tag, value) pairs. The following  $4*(N-1)$  bytes are offsets, each a uint32. The last  $4*N$  bytes in the header are tags.

Offsets refer to the positions of the values in the message values section. All offsets MUST be multiples of four and placed in increasing order. The first post-header byte is at offset 0. The offset array is considered to have a not explicitly encoded value of 0 as its zeroth entry. The value associated with the  $i$ th tag begins at offset[i] and ends at offset[i+1]-1, with the exception of the last value which ends at the end of the message. Values may have zero length.

Tags MUST be listed in the same order as the offsets of their values and MUST also be sorted in ascending order by numeric value. A tag MUST NOT appear more than once in a header.

## 6. Protocol Details

As described in Section 3, clients initiate time synchronization by sending requests containing a nonce to servers who send signed time responses in return. RoughTime packets can be sent between clients and servers either as UDP datagrams or via TCP streams. Servers **SHOULD** support the UDP transport mode, while TCP transport is **OPTIONAL**.

A RoughTime packet **MUST** be formatted according to Figure 2 and as described here. The first field is a uint64 with the value 0x4d49544847554f52 ("ROUGHTIM" in ASCII). The second field is a uint32 and contains the length of the third field. The third and last field contains a RoughTime message as specified in Section 5.1.

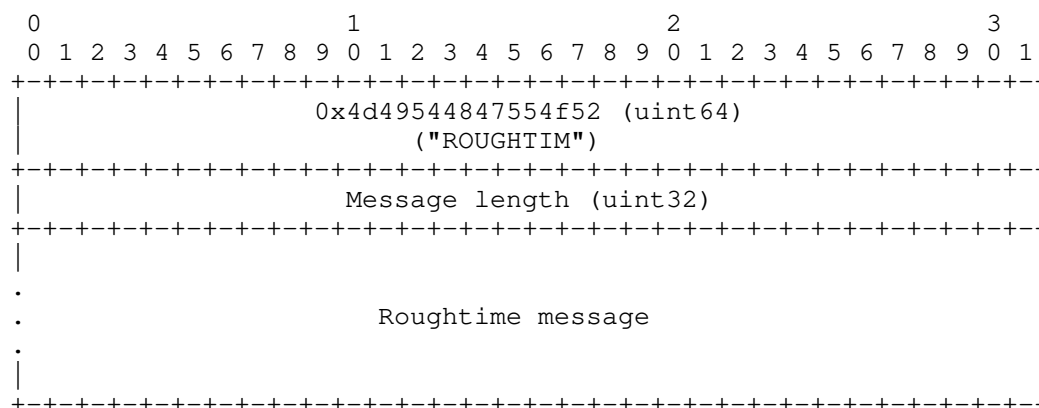


Figure 2: RoughTime Packet Format

RoughTime request and response packets **MUST** be transmitted in a single datagram when the UDP transport mode is used. Setting the packet's don't fragment bit [RFC0791] is **OPTIONAL** in IPv4 networks.

Multiple requests and responses can be exchanged over an established TCP connection. Clients **MAY** send multiple requests at once and servers **MAY** send responses out of order. The connection **SHOULD** be closed by the client when it has no more requests to send and has received all expected responses. Either side **SHOULD** close the connection in response to synchronization, format, implementation-defined timeouts, or other errors.

All requests and responses **MUST** contain the VER tag. It contains a list of one or more uint32 version numbers. The version of RoughTime specified by this memo has version number 1.

NOTE TO RFC EDITOR: remove this paragraph before publication. For testing drafts of this memo, a version number of 0x80000000 plus the draft number is used.

## 6.1. Requests

A request **MUST** contain the tags VER and NONC. Tags other than NONC and VER **SHOULD** be ignored by the server. A future version of this protocol may mandate additional tags in the message and assign them semantic meaning.

The size of the request message **SHOULD** be at least 1024 bytes when the UDP transport mode is used. To attain this size the PAD tag **SHOULD** be added to the message. Its value **SHOULD** be all zeros. Responding to requests shorter than 1024 bytes is **OPTIONAL** and servers **MUST NOT** send responses larger than the requests they are replying to.

### 6.1.1. VER

In a request, the VER tag contains a list of versions. The VER tag **MUST** include at least one RoughTime version supported by the client. The client **MUST** ensure that the version numbers and tags included in the request are not incompatible with each other or the packet contents.

### 6.1.2. NONC

The value of the NONC tag is a 32 byte nonce. It **SHOULD** be generated in a manner indistinguishable from random. BCP 106 contains specific guidelines regarding this [RFC4086].

## 6.2. Responses

A response **MUST** contain the tags SIG, VER, NONC, PATH, SREP, CERT, and INDX.

### 6.2.1. SIG

In general, a SIG tag value is a 64 byte Ed25519 signature [RFC8032] over a concatenation of a signature context ASCII string and the entire value of a tag. All context strings **MUST** include a terminating zero byte.

The SIG tag in the root of a response **MUST** be a signature over the SREP value using the public key contained in CERT. The context string **MUST** be "RoughTime v1 response signature".

#### 6.2.2. VER

In a response, the VER tag MUST contain a single version number. It SHOULD be one of the version numbers supplied by the client in its request. The server MUST ensure that the version number corresponds with the rest of the packet contents.

#### 6.2.3. NONC

The NONC tag MUST contain the nonce of the message being responded to.

#### 6.2.4. PATH

The PATH tag value MUST be a multiple of 32 bytes long and represent a path of 32 byte hash values in the Merkle tree used to generate the ROOT value as described in Section 6.3. In the case where a response is prepared for a single request and the Merkle tree contains only the root node, the size of PATH MUST be zero.

#### 6.2.5. SREP

The SREP tag contains a time response. Its value MUST be a RoughTime message with the tags ROOT, MIDP, and RADI. The server MAY include any of the tags DUT1, DTAI, and LEAP in the contents of the SREP tag.

The ROOT tag MUST contain a 32 byte value of a Merkle tree root as described in Section 6.3.

The MIDP tag value MUST be timestamp of the moment of processing.

The RADI tag value MUST be a uint32 representing the server's estimate of the accuracy of MIDP in microseconds. Servers MUST ensure that the true time is within (MIDP-RADI, MIDP+RADI) at the time they transmit the response message.

The DUT1 tag value MUST be an int32 indicating the predicted difference between UT1 and UTC (UT1 - UTC) in milliseconds as given by the International Earth Rotation and Reference Systems Service (IERS).

The DTAI tag value MUST be an int32 indicating the current difference between International Atomic Time (TAI) and UTC (TAI - UTC) in milliseconds as published in the International Bureau of Weights and Measures' (BIPM) Circular T.

The LEAP tag MUST contain zero or more int32 values, each representing a past or future leap second event. Positive values represent the addition of a second and negative values represent the removal of a second. The absolute value represents the MJD of the day that begins immediately after the leap second event.

By way of illustration, there was a leap second 31 December 2016 23:59:60. This event would be represented by the tag with numeric value 57754. The positive sign represents that there was an additional second inserted, the numeric value indicates 1 January 2017, the following day that began at midnight after the addition.

The leap second events MUST be sorted in reverse chronological order and the first item MUST be the last (past or future) leap second event that the server knows about. A LEAP tag with zero int32 values indicates that the server does not hold any updated leap second information.

#### 6.2.6. CERT

The CERT tag contains a public-key certificate signed with the server's long-term key. Its value is a RoughTime message with the tags DELE and SIG, where SIG is a signature over the DELE value. The context string used to generate SIG MUST be "RoughTime v1 delegation signature--".

The DELE tag contains a delegated public-key certificate used by the server to sign the SREP tag. Its value is a RoughTime message with the tags MINT, MAXT, and PUBK. The purpose of the DELE tag is to enable separation of a long-term public key from keys on devices exposed to the public Internet.

The MINT tag is the minimum timestamp for which the key in PUBK is trusted to sign responses. MIDP MUST be more than or equal to MINT for a response to be considered valid.

The MAXT tag is the maximum timestamp for which the key in PUBK is trusted to sign responses. MIDP MUST be less than or equal to MAXT for a response to be considered valid.

The PUBK tag contains a temporary 32 byte Ed25519 public key which is used to sign the SREP tag.

#### 6.2.7. INDX

The INDX tag value is a uint32 determining the position of NONC in the Merkle tree used to generate the ROOT value as described in Section 6.3.

### 6.3. The Merkle Tree

A Merkle tree is a binary tree where the value of each non-leaf node is a hash value derived from its two children. The root of the tree is thus dependent on all leaf nodes.

In RoughTime, each leaf node in the Merkle tree represents the nonce in one request. Leaf nodes are indexed left to right, beginning with zero.

The values of all nodes are calculated from the leaf nodes and up towards the root node using the first 32 bytes of the output of the SHA-512 hash algorithm [SHS]. For leaf nodes, the byte 0x00 is prepended to the nonce before applying the hash function. For all other nodes, the byte 0x01 is concatenated with first the left and then the right child node value before applying the hash function.

The value of the Merkle tree's root node is included in the ROOT tag of the response.

The index of a request's nonce node is included in the INDX tag of the response.

The values of all sibling nodes in the path between a request's nonce node and the root node is stored in the PATH tag so that the client can reconstruct and validate the value in the ROOT tag using its nonce. These values are each 32 bytes and are stored one after the other with no additional padding or structure. The order in which they are stored is described in Section 6.3.1

#### 6.3.1. Root Value Validity Check Algorithm

We describe how to compute the hash of the Merkle tree from the values in the tags PATH, INDX, and NONC. Our algorithm maintains a current hash value. The bits of INDX are ordered from least to most significant in this algorithm.

At initialization hash is set to  $H(0x00 || \text{nonce})$ .

If no more entries remain in PATH the current hash is the hash of the Merkle tree. All remaining bits of INDX must be zero.

Otherwise let node be the next 32 bytes in PATH. If the current bit in INDX is 0 then  $\text{hash} = H(0x01 || \text{node} || \text{hash})$ , else  $\text{hash} = H(0x01 || \text{hash} || \text{node})$ .

#### 6.4. Validity of Response

A client MUST check the following properties when it receives a response. We assume the long-term server public key is known to the client through other means.

- \* The signature in CERT was made with the long-term key of the server.
- \* The DELE timestamps and the MIDP value are consistent.
- \* The INDX and PATH values prove NONC was included in the Merkle tree with value ROOT using the algorithm in Section 6.3.1.
- \* The signature of SREP in SIG validates with the public key in DELE.

A response that passes these checks is said to be valid. Validity of a response does not prove the time is correct, but merely that the server signed it, and thus promises that it began to compute the signature at a time in the interval (MIDP-RADI, MIDP+RADI).

#### 7. Integration Into NTP

We assume that there is a bound PHI on the frequency error in the clock on the machine. Given a measurement taken at a local time  $t$ , we know the true time is in  $(t - \text{delta} - \text{sigma}, t - \text{delta} + \text{sigma})$ . After  $d$  seconds have elapsed we know the true time is within  $(t - \text{delta} - \text{sigma} - d * \text{PHI}, t - \text{delta} + \text{sigma} + d * \text{PHI})$ . A simple and effective way to mix with NTP or PTP discipline of the clock is to trim the observed intervals in NTP to fit entirely within this window or reject measurements that fall too far outside. This assumes time has not been stepped. If the NTP process decides to step the time, it MUST use Roughtime to ensure the new truetime estimate that will be stepped to is consistent with the true time.

Should this window become too large, another Roughtime measurement is called for. The definition of "too large" is implementation defined.

Implementations MAY use other, more sophisticated means of adjusting the clock respecting Roughtime information. Other applications such as X.509 verification may wish to

## 8. Grease

Servers MAY send back a fraction of responses that are syntactically invalid or contain invalid signatures as well as incorrect times. Clients MUST properly reject such responses. Servers MUST NOT send back responses with incorrect times and valid signatures. Either signature MAY be invalid for this application.

## 9. RoughTime Servers

NOTE TO RFC EDITOR: remove this section before publication.

The below list contains a list of servers with their public keys in Base64 format. These servers may implement older versions of this specification.

address:           roughTime.cloudflare.com  
port:              2002  
long-term key: gD63hSj3ScS+wuOeGrubXlq35N1c5Lby/S+T7MNTjxo=

address:           roughTime.int08h.com  
port:              2002  
long-term key: AW5uAoTSTDfG5NfY1bTh08GUnOqlRb+HVhbJ3ODJvsE=

address:           roughTime.sandbox.google.com  
port:              2002  
long-term key: etPaaIxcBMY1oUeGpwvPMCJMw1RVNxv51KK/tktoJTQ=

address:           roughTime.se  
port:              2002  
long-term key: S3AzfZJ5CjSdkJ21ZJGbxqdYP/SoE8fXKY0+aicsehI=

## 10. Acknowledgements

Thomas Peterson corrected multiple nits. Peter Loethberg, Tal Mizrahi, Ragnar Sundblad, Kristof Teichel, and the other members of the NTP working group contributed comments and suggestions.

## 11. IANA Considerations

### 11.1. Service Name and Transport Protocol Port Number Registry

IANA is requested to allocate the following entry in the Service Name and Transport Protocol Port Number Registry [RFC6335]:

Service Name: RoughTime

Transport Protocol: tcp,udp

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: Roughtime time synchronization

Reference: [[this memo]]

Port Number: [[TBD1]], selected by IANA from the User Port range

### 11.2. Roughtime Version Registry

IANA is requested to create a new registry entitled "Roughtime Version Registry". Entries shall have the following fields:

Version ID (REQUIRED): a 32-bit unsigned integer

Version name (REQUIRED): A short text string naming the version being identified.

Reference (REQUIRED): A reference to a relevant specification document.

The policy for allocation of new entries SHOULD be: IETF Review.

The initial contents of this registry shall be as follows:

Version ID	Version name	Reference
0x0	Reserved	[[this memo]]
0x1	Roughtime version 1	[[this memo]]
0x2-0x7fffffff	Unassigned	
0x80000000-0xffffffff	Reserved for Private or Experimental use	[[this memo]]

Table 2: Roughtime version assignments.

### 11.3. Roughtime Tag Registry

IANA is requested to create a new registry entitled "Roughtime Tag Registry". Entries SHALL have the following fields:

Tag (REQUIRED): A 32-bit unsigned integer in hexadecimal format.

ASCII Representation (OPTIONAL): The ASCII representation of the tag in accordance with Section 5.1.4 of this memo, if applicable.

Reference (REQUIRED): A reference to a relevant specification document.

The policy for allocation of new entries in this registry SHOULD be: Specification Required.

The initial contents of this registry SHALL be as follows:

Tag	ASCII Representation	Reference
0x00444150	PAD	[[this memo]]
0x00474953	SIG	[[this memo]]
0x00524556	VER	[[this memo]]
0x31545544	DUT1	[[this memo]]
0x434e4f4e	NONC	[[this memo]]
0x454c4544	DELE	[[this memo]]
0x48544150	PATH	[[this memo]]
0x49415444	DTAI	[[this memo]]
0x49444152	RADI	[[this memo]]
0x4b425550	PUBK	[[this memo]]
0x5041454c	LEAP	[[this memo]]
0x5044494d	MIDP	[[this memo]]
0x50455253	SREP	[[this memo]]
0x544e494d	MINT	[[this memo]]
0x544f4f52	ROOT	[[this memo]]
0x54524543	CERT	[[this memo]]
0x5458414d	MAXT	[[this memo]]
0x58444e49	INDX	[[this memo]]

Table 3: RoughTime tags.

## 12. Security Considerations

Since the only supported signature scheme, Ed25519, is not quantum resistant, the RoughTime version described in this memo will not survive the advent of quantum computers.

Maintaining a list of trusted servers and adjudicating violations of the rules by servers is not discussed in this document and is essential for security. RoughTime clients MUST regularly update their view of which servers are trustworthy in order to benefit from the detection of misbehavior.

Validating timestamps made on different dates requires knowledge of leap seconds in order to calculate time intervals correctly.

Servers carry out a significant amount of computation in response to clients, and thus may experience vulnerability to denial of service attacks.

This protocol does not provide any confidentiality. Given the nature of timestamps such impact is minor.

The compromise of a PUBK's private key, even past MAXT, is a problem as the private key can be used to sign invalid times that are in the range MINT to MAXT, and thus violate the good behavior guarantee of the server.

Servers MUST NOT send response packets larger than the request packets sent by clients, in order to prevent amplification attacks.

### 13. Privacy Considerations

This protocol is designed to obscure all client identifiers. Servers necessarily have persistent long-term identities essential to enforcing correct behavior.

Generating nonces in a nonrandom manner can cause leaks of private data or enable tracking of clients as they move between networks.

### 14. References

#### 14.1. Normative References

[ITU-R\_TF.457-2]

ITU-R, "Use of the Modified Julian Date by the Standard-Frequency and Time-Signal Services", ITU-R Recommendation TF.457-2, October 1997.

[ITU-R\_TF.460-6]

ITU-R, "Standard-Frequency and Time-Signal Emissions", ITU-R Recommendation TF.460-6, February 2002.

- [RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <<https://www.rfc-editor.org/info/rfc20>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/info/rfc8032>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [SHS] National Institute of Standards and Technology, "Secure Hash Standard", DOI 10.6028/NIST.FIPS.180-4, FIPS 180-4, August 2015, <<https://doi.org/10.6028/NIST.FIPS.180-4>>.

#### 14.2. Informative References

- [Autokey] Rottger, S., "Analysis of the NTP Autokey Procedures", 2012, <[https://zero-entropy.de/autokey\\_analysis.pdf](https://zero-entropy.de/autokey_analysis.pdf)>.
- [DelayAttacks] Mizrahi, T., "A Game Theoretic Analysis of Delay Attacks Against Time Synchronization Protocols", DOI 10.1109/ISPCS.2012.6336612, 2012, <<https://ieeexplore.ieee.org/document/6336612>>.
- [MCBG] Malhotra, A., Cohen, I., Brakke, E., and S. Goldberg, "Attacking the Network Time Protocol", 2015, <<https://eprint.iacr.org/2015/1020>>.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<https://www.rfc-editor.org/info/rfc768>>.
- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<https://www.rfc-editor.org/info/rfc791>>.

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.
- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.
- [RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <<https://www.rfc-editor.org/info/rfc7384>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8573] Malhotra, A. and S. Goldberg, "Message Authentication Code for the Network Time Protocol", RFC 8573, DOI 10.17487/RFC8573, June 2019, <<https://www.rfc-editor.org/info/rfc8573>>.
- [RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <<https://www.rfc-editor.org/info/rfc8915>>.

## Appendix A. Terms and Abbreviations

ASCII American Standard Code for Information Interchange

IANA Internet Assigned Numbers Authority

JSON JavaScript Object Notation [RFC8259]

MJD Modified Julian Date

NTP Network Time Protocol [RFC5905]

NTS Network Time Security [RFC8915]

TAI International Atomic Time (Temps Atomique International)  
[ITU-R\_TF.460-6]

TCP Transmission Control Protocol [RFC0793]

UDP User Datagram Protocol [RFC0768]

UT Universal Time [ITU-R\_TF.460-6]

UTC Coordinated Universal Time [ITU-R\_TF.460-6]

#### Authors' Addresses

Aanchal Malhotra  
Boston University  
111 Cummington Mall  
Boston, MA 02215  
United States of America

Email: aanchal4@bu.edu

Adam Langley  
Google

Email: agl@google.com

Watson Ladd  
Cloudflare  
101 Townsend St  
San Francisco, CA 94107  
United States of America

Email: watsonbladd@gmail.com

Marcus Dansarie

Email: marcus@dansarie.se  
URI: <https://orcid.org/0000-0001-9246-0263>

Internet Engineering Task Force  
Internet-Draft  
Intended status: Informational  
Expires: August 25, 2021

W. Ladd  
Cloudflare  
M. Dansarie  
February 21, 2021

Roughtime Ecosystem  
draft-ietf-ntp-roughtime-ecosystem-00

## Abstract

This document specifies the roles of Roughtime validators, clients, and servers in providing a ecosystem for secure time.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 25, 2021.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Chaining in roughtime . . . . .	2
3. Impeachment . . . . .	2
4. Serialization of chains . . . . .	3
5. Submission API . . . . .	3
6. Viewing Reports . . . . .	3
7. Trust Anchors and Policies . . . . .	3
8. Normative References . . . . .	3
Authors' Addresses . . . . .	3

## 1. Introduction

The Roughtime protocol enables servers to provide cryptographic proof of the times requests were made. This enables clients to expose cheating by servers. This document describes how these proofs are serialized and verified, as well as APIs to access and submit reports of malfeasance in an automated manner.

## 2. Chaining in roughtime

Two responses are chained if the NONC field of the second is SHA-512(blinder || first) where blinder is a 64 byte value. Blinder MUST be generated uniformly at random to prevent tracking. The first response is serialized as a roughtime message. The first response is chained to the second.

A chain is a sequence of messages where each message is chained to the one before. Every contiguous subsequence of a chain is a chain.

## 3. Impeachment

For each index  $i$ , let  $m_i$  denote the timestamp of the response,  $r_i$  the radius around it. Then we have  $m_i - r_i$  the earliest actual time at which the response could have been generated, and  $m_i + r_i$  the latest actual time at which the response could have been generated.

If all requests are generated honestly  $m_i + r_i < m_{i+j} - r_{i+j}$  holds for all indices  $i$  and positive numbers  $j$ . A failure of this relation to hold demonstrates that at least one of the responses was generated incorrectly.

The more distinct servers and responses that are mutually consistent except for the questionable response, the more likely a failure of the generator of the erroneous response is.

#### 4. Serialization of chains

TODO

#### 5. Submission API

#### 6. Viewing Reports

#### 7. Trust Anchors and Policies

A trust anchor is any distributor of a list of trusted servers. It is RECOMMENDED that trust anchors subscribe to a common public forum where evidence of malfeasance may be shared and discussed. Trust anchors SHOULD subscribe to a zero-tolerance policy: any generation of incorrect timestamps will result in removal. To enable this trust anchors SHOULD list a wide variety of servers so the removal of a server does not result in operational issues for clients. Clients SHOULD attempt to detect malfeasance and report it as discussed in this document.

Because only a single Roughtime server is required for successful synchronization, Roughtime does not have the incentive problems that have prevented effective enforcement of discipline on the web PKI.

#### 8. Normative References

[I-D.ietf-ntp-roughtime]

Malhotra, A., Langley, A., and W. Ladd, "Roughtime",  
draft-ietf-ntp-roughtime-03 (work in progress), August  
2020.

#### Authors' Addresses

Watson Ladd  
Cloudflare  
101 Townsend St  
San Francisco  
USA

Email: watsonbladd@gmail.com

Marcus Dansarie  
Sweden

Email: marcus@dansarie.se  
URI: <https://orcid.org/0000-0001-9246-0263>

Internet Engineering Task Force  
Internet-Draft  
Intended status: Informational  
Expires: 18 March 2022

W. Ladd  
Cloudflare  
M. Dansarie  
September 2021

Roughtime Ecosystem  
draft-ietf-ntp-roughtime-ecosystem-01

## Abstract

This document specifies the roles of Roughtime validators, clients, and servers in providing a ecosystem for secure time.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 5 March 2022.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Chaining in roughtime . . . . .	2
3. Impeachment . . . . .	2
4. Serialization of chains . . . . .	3
5. Submission API . . . . .	3
6. Viewing Reports . . . . .	3
7. Trust Anchors and Policies . . . . .	3
8. Normative References . . . . .	3
Authors' Addresses . . . . .	3

## 1. Introduction

The Roughtime protocol enables servers to provide cryptographic proof of the times requests were made. This enables clients to expose cheating by servers. This document describes how these proofs are serialized and verified, as well as APIs to access and submit reports of malfeasance in an automated manner.

## 2. Chaining in roughtime

Two responses are chained if the NONC field of the second is `SHA-512(blinder || first)` where blinder is a 64 byte value. Blinder MUST be generated uniformly at random to prevent tracking. The first response is serialized as a roughtime message. The first response is chained to the second.

A chain is a sequence of messages where each message is chained to the one before. Every contiguous subsequence of a chain is a chain.

## 3. Impeachment

For each index  $i$ , let  $m_i$  denote the timestamp of the response,  $r_i$  the radius around it. Then we have  $m_i - r_i$  the earliest actual time at which the response could have been generated, and  $m_i + r_i$  the latest actual time at which the response could have been generated.

If all requests are generated honestly  $m_i + r_i < m_{i+j} - r_{i+j}$  holds for all indices  $i$  and positive numbers  $j$ . A failure of this relation to hold demonstrates that at least one of the responses was generated incorrectly.

The more distinct servers and responses that are mutually consistent except for the questionable response, the more likely a failure of the generator of the erroneous response is.

#### 4. Serialization of chains

TODO

#### 5. Submission API

#### 6. Viewing Reports

#### 7. Trust Anchors and Policies

A trust anchor is any distributor of a list of trusted servers. It is RECOMMENDED that trust anchors subscribe to a common public forum where evidence of malfeasance may be shared and discussed. Trust anchors SHOULD subscribe to a zero-tolerance policy: any generation of incorrect timestamps will result in removal. To enable this trust anchors SHOULD list a wide variety of servers so the removal of a server does not result in operational issues for clients. Clients SHOULD attempt to detect malfeasance and report it as discussed in this document.

Because only a single Roughtime server is required for successful synchronization, Roughtime does not have the incentive problems that have prevented effective enforcement of discipline on the web PKI.

#### 8. Normative References

[I-D.ietf-ntp-roughtime]

Malhotra, A., Langley, A., Ladd, W., and M. Dansarie,  
"Roughtime", Work in Progress, Internet-Draft, draft-ietf-  
ntp-roughtime-05, 24 May 2021,  
<<https://www.ietf.org/archive/id/draft-ietf-ntp-roughtime-05.txt>>.

#### Authors' Addresses

Watson Ladd  
Cloudflare  
101 Townsend St  
San Francisco,  
United States of America

Email: [watsonbladd@gmail.com](mailto:watsonbladd@gmail.com)

Marcus Dansarie  
Sweden

Email: [marcus@dansarie.se](mailto:marcus@dansarie.se)

URI: <https://orcid.org/0000-0001-9246-0263>

Network Time Protocol  
Internet-Draft  
Intended status: Standards Track  
Expires: 24 August 2022

M. Langer  
R. Bermbach  
Ostfalia University  
20 February 2022

NTS4PTP - Key Management System for the Precision Time Protocol Based on  
the Network Time Security Protocol  
draft-langer-ntp-nts-for-ntp-03

## Abstract

This document defines a key management service for automatic key management for the integrated security mechanism (prong A) of IEEE Std 1588[TM]-2019 (PTPv2.1) described there in Annex P. It implements a key management for the immediate security processing approach and offers a security solution for all relevant PTP modes. The key management service for PTP is based on and extends the NTS Key Establishment protocol defined in IETF RFC 8915 for securing NTP, but works completely independent from NTP.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 24 August 2022.

## Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components

extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Notational Conventions . . . . .	3
2. Key Management Using Network Time Security . . . . .	3
2.1. Principle Key Distribution Mechanism . . . . .	5
2.1.1. NTS Message Exchange for Group-based Approach . . . . .	8
2.1.2. NTS Message Exchange for the Ticket-based Approach . . . . .	10
2.2. General Topics . . . . .	14
2.2.1. Key Update Process . . . . .	14
2.2.2. Key Generation . . . . .	19
2.2.3. Time Information of the NTS-KE server . . . . .	19
2.2.4. Certificates . . . . .	19
2.2.5. Upfront Configuration . . . . .	20
2.2.5.1. Security Parameters . . . . .	20
2.2.5.2. Key Lifetimes . . . . .	21
2.2.5.3. Certificates . . . . .	21
2.2.5.4. Authorization . . . . .	21
2.2.5.5. Transparent Clocks . . . . .	22
2.2.5.6. Start-up considerations . . . . .	23
2.3. Overview of NTS Messages and their Structure for Use with PTP . . . . .	23
2.3.1. PTP Key Request Message . . . . .	25
2.3.2. PTP Key Response Message . . . . .	26
2.3.3. PTP Registration Request Message . . . . .	27
2.3.4. PTP Registration Response Message . . . . .	28
2.3.5. PTP Registration Revoke Message . . . . .	30
2.3.6. Heartbeat Message . . . . .	31
3. NTS Messages for PTP . . . . .	32
3.1. NTS Message Types . . . . .	34
3.2. NTS Records . . . . .	38
3.2.1. AEAD Algorithm Negotiation . . . . .	38
3.2.2. Association Mode . . . . .	40
3.2.3. Current Parameters . . . . .	43
3.2.4. End of Message . . . . .	45
3.2.5. Error . . . . .	45
3.2.6. Heartbeat Timeout . . . . .	46
3.2.7. Next Parameters . . . . .	47
3.2.8. NTS Next Protocol Negotiation . . . . .	48
3.2.9. NTS Message Type . . . . .	49
3.2.10. PTP Time Server . . . . .	50
3.2.11. Security Association . . . . .	51
3.2.12. Source PortIdentity . . . . .	53
3.2.13. Status . . . . .	54
3.2.14. Supported MAC Algorithms . . . . .	56

3.2.15. Ticket . . . . .	58
3.2.16. Ticket Key . . . . .	60
3.2.17. Ticket Key ID . . . . .	60
3.2.18. Validity Period . . . . .	61
4. Additional Mechanisms . . . . .	63
4.1. AEAD Operation . . . . .	63
4.2. SA/SP Management . . . . .	65
5. New TICKET TLV for PTP Messages . . . . .	67
6. AUTHENTICATION TLV Parameters . . . . .	69
7. IANA Considerations . . . . .	70
8. Security Considerations . . . . .	70
9. Acknowledgements . . . . .	70
10. References . . . . .	70
10.1. Normative References . . . . .	70
10.2. Informative References . . . . .	71
Authors' Addresses . . . . .	72

## 1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 2. Key Management Using Network Time Security

In its annex P the IEEE Std 1588-2019 ([IEEE1588-2019], Precision Time Protocol version 2.1, PTPv2.1) defines a comprehensive PTP security concept based on four prongs (A to D). Prong A incorporates an immediate security processing approach and specifies in section 16.14 an extension to secure PTP messages by means of an AUTHENTICATION TLV (AuthTLV) containing an Integrity Check Value (ICV). For PTP instances to use the securing mechanism, a respective key needs to be securely distributed among them. Annex P gives requirements for such a key management system and mentions potential candidates without further specification, but allows other solutions as long as they fulfill those requirements.

This document defines such a key management service for automatic key management for the immediate security processing in prong A. The solution [Langer\_et\_al.\_2020] is based on and expands the NTS Key Establishment protocol defined in IETF RFC 8915 [RFC8915] for securing NTP, but works completely independent from NTP.

Many networks include both, PTP and NTP at the same time. Furthermore, many time server appliances that are capable of acting as the Grandmaster of a PTP network are also capable of acting as an

NTP server. For these reasons, it is likely to be easier both, for the time server manufacturer and the network operator, if PTP and NTP use a key management system based on the same technology. The Network Time Security (NTS) protocol was specified by the Internet Engineering Task Force (IETF) to protect the integrity of NTP messages [RFC8915]. Its NTS Key Establishment sub-protocol is secured by the Transport Layer Security (TLS 1.3, IETF RFC 8446 [RFC8446]) mechanism. TLS is used to protect numerous popular network protocols, so it is present in many networks. For example, HTTPS, the predominant secure web protocol uses TLS for security. Since many PTP capable network appliances have management interfaces based on HTTPS, the manufacturers are already implementing TLS.

Though the key management for PTP is based on the NTS Key Establishment (NTS-KE) protocol for NTP, it works completely independent of NTP. The key management system uses the procedures described in IETF RFC 8915 for the NTS-KE and expands it with new NTS messages for PTP. It may be applied in a Key Establishment server (NTS-KE server) that already manages NTP but can also be operated only handling KE for PTP. Even when the PTP network is isolated from the Internet, a Key Establishment server can be installed in that network providing the PTP instances with necessary key and security parameters.

The NTS-KE server may often be implemented as a separate unit. It also may be collocated with a PTP instance, e.g., the Grandmaster. In the latter case communication between the NTS-KE server program and the PTP instance program needs to be implemented in a secure way if TLS communication (e.g., via local host) is not or cannot be used.

Using the expanded NTS Key Establishment protocol for the NTS key management for PTP, NTS4PTP provides two principle approaches specified in this document.

1. Group-based approach (GrBA, multicast)

- \* definition of one or more security groups in the PTP network,
- \* very suitable for PTP multicast mode and mixed multicast/unicast mode,
- \* suitable for unicast mode in small subgroups of very few participants (Group-of-2, Go2) but poor scaling and more administration work,

2. Ticket-based approach (TiBA, unicast)

- \* secured (end-to-end) PTP unicast communication between a PTP requester and grantor,
- \* no group binding necessary,

- \* very suitable for native PTP unicast mode, because of good scaling,
- \* a bit more complex NTS message handling.

For these modes, the NTS key management for PTP defines six new NTS messages which will be introduced in the sections to come:

- \* PTP Key Request message (see Section 2.3.1)
- \* PTP Key Response message (see Section 2.3.2)
- \* PTP Registration Request message (see Section 2.3.3)
- \* PTP Registration Response message (see Section 2.3.4)
- \* PTP Registration Revoke message (see Section 2.3.5)
- \* Heartbeat message (see Section 2.3.6)

This document describes the structure and usage of the two approaches GrBA and TiBA in their application as a key management system for the integrated security mechanism (prong A) of IEEE Std 1588-2019. Section 2.1 starts with a description of the principle key distribution mechanism, continues with details of the various group-based options (Section 2.1.1) and the ticket-based unicast mode (Section 2.1.2) before it ends with more general topics in Section 2.2 for example the key update process and finally an overview of the newly defined NTS messages in Section 2.3. Section 3 gives all the details necessary to construct all records forming the particular NTS messages. Section 5 depicts details of a TICKET TLV needed to transport encrypted security information in PTP unicast requests. The following Section 6 mentions specific parameters used in the PTP AUTHENTICATION TLV when working with the NTS4PTP key management system. Section 7 and Section 8 discuss IANA respectively security considerations.

## 2.1. Principle Key Distribution Mechanism

A PTP instance requests a key from the server referred to as the Key Establishment server, or NTS-KE server using the NTS-KE protocol defined in [RFC8915], see Section 1.3. Figure 1 describes the principle sequence which can be used for PTP multicast as well as PTP unicast operation.

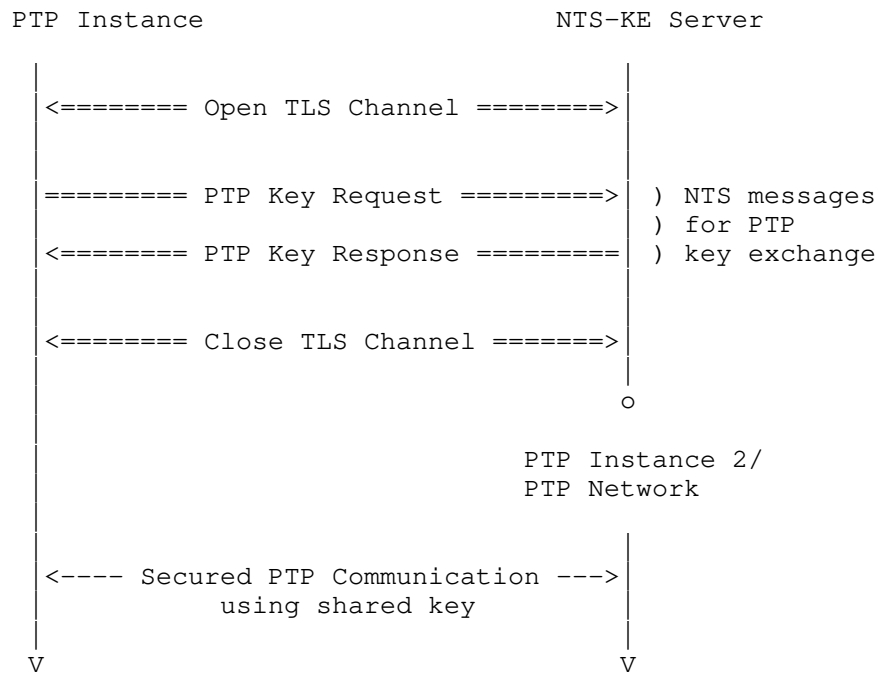


Figure 1: NTS key distribution sequence

The PTP instance client connects to the NTS-KE server on the NTS TCP port (port number 4460). Then both parties perform a TLS handshake to establish a TLS 1.3 communication channel. No earlier TLS versions are allowed. The details of the TLS handshake are specified in IETF RFC 8446 [RFC8446].

Implementations must conform to the rules stated in Section 3 "TLS Profile for Network Time Security" of IETF RFC 8915 [RFC8915]:

```

_"Network Time Security makes use of TLS for NTS key
establishment._
_Since the NTS protocol is new as of this publication, no
backward-compatibility concerns exist to justify using obsolete,
insecure, or otherwise broken TLS features or versions._
_Implementations MUST conform with RFC 7525_ [RFC7525]_or with a
later revision of BCP 195._
_Implementations MUST NOT negotiate TLS versions earlier than
1.3_[RFC8446]_and MAY refuse to negotiate any TLS version that has
been superseded by a later supported version._

```

`_Use of the Application-Layer Protocol Negotiation  
Extension_[RFC7301]_is integral to NTS, and support for it is  
REQUIRED for interoperability ... "_`

The client starts the TLS handshake with a 'Client Hello' message that must contain two TLS extensions. The first extension is the Application Layer Protocol Negotiation [RFC7301] (ALPN with "ntske/1", which refers to the NTS Key Establishment as the subsequent protocol.) The second extension is the Post-Handshake Client Authentication, which the client uses to signal the TLS server that the client certificate can be requested after the TLS handshake. Afterwards, the client authenticates the NTS-KE server using the root CA certificate or by means of the Online Certificate Status Protocol (OCSP). Both, client and server agree on the cipher suite and then establish a secured channel that ensures authenticity, integrity and confidentiality for subsequent messages. In the process, the NTS-KE server acknowledges the ALPN and expects a message from the NTS-KE protocol.

Thus, the TLS handshake accomplishes the following:

- \* Negotiation of TLS version (only TLS 1.3 allowed), and
- \* negotiation of the cipher suite for the TLS session, and
- \* authentication of the TLS server (equivalent to the NTS-KE server) using a digital X.509 certificate,
- \* and the encryption of the subsequent information exchange between the TLS communication partners.

TLS is a layer five protocol that runs on TCP over IP. Therefore, PTP implementations that support NTS-based key management need to support TCP and IP (at least on a separate management port).

Once the TLS session is established, the PTP instance will ask for a PTP key as well as the associated security parameters using the new NTS message PTP Key Request (see Section 2.3.1). Then the server requests the client's X.509 certificate (via TLS Certificate Request) and verifies it upon receipt. In NTS for NTP this was unnecessary, in NTS4PTP the clients MUST be authenticated, too. The NTS application of the NTS-KE server will respond with a PTP Key Response message (see Section 2.3.2). If no delivery of security data is possible for whatever reason, the PTP Key Response message contains a respective error code. All messages are constructed from specific records as described in Section 3.2.

When the PTP Key Request message was responded with a PTP Key Response, the TLS session will be closed with a 'close notify' TLS alert from both parties, the PTP instance and the key server.

With the key and other information received, the PTP instance can take part in the secured PTP communication in the different modes of operation.

After the reception of the first set of security parameters the PTP instance may resume the TLS session according to IETF RFC 8446 [RFC8446], Section 4.6.1, allowing the PTP instance to skip the TLS version and algorithm negotiations. If TLS Session Resumption ([RFC8446], Section 2.2) is used and supported by the NTS-KE server, a suitable lifetime (max. 24 hrs) for the TLS session key must be defined to not open the TLS connection for security threats. If the NTS-KE server does not support TLS resumption, a full TLS handshake must be performed.

As the TLS session provides authentication, but not authorization additional means have to be used for the latter (see Section 2.2.5.4).

As mentioned above, the NTS key management for PTP supports two principle methods, the group-based approach (GrBA) and the ticket-based approach (TiBA) which are described in the following sections below.

#### 2.1.1. NTS Message Exchange for Group-based Approach

As described in Section 2.1, a PTP instance wanting to join a secured PTP communication in the group-based modes contacts the NTS-KE server starting the establishment of a secured TLS connection using the NTS-KE protocol (ALPN: ntske/1). Then, the client continues with a PTP Key Request message, asking for a specific group (see Section 2.3.1) as shown in Figure 2. After receiving the message, the NTS-KE server requests the client's certificate and performs an authorization check. The NTS-KE server then replies with a PTP Key Response message (see Section 2.3.2) with all the necessary data to join the group communication. Else, it contains a respective error code if the PTP instance is not allowed to join the group. This procedure is necessary for all parties, which are or will be members of that PTP group including the Grandmaster and other special participants, e.g., Transparent Clocks. As mentioned above, this not only applies to multicast mode but also to mixed multicast/unicast mode (former hybrid mode) where the explicit unicast communication uses the multicast group key received from the NTS-KE server. The group number for both modes is primarily generated by a concatenation of the PTP domain number and the PTP profile identifier (sdoId), as described in Section 3.2.2.

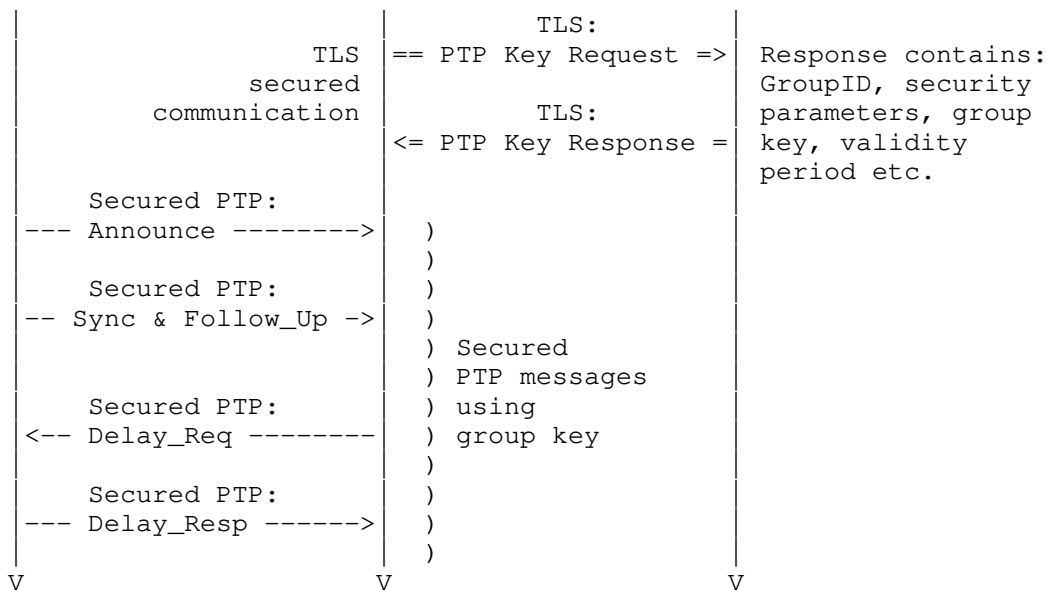
Additionally, besides multicast and mixed multicast/unicast mode, a group of two (or few more) PTP instances can be configured, practically implementing a special group-based unicast communication mode, the group-of-2 (Go2) mode.

#### Secured

PTP Network

PTP Instance

NTS-KE Server



Legend:

TLS: Authenticated & encrypted  
 =====> TLS communication

Secured PTP: Group key-authenticated  
 -----> PTP communication

Figure 2: Message exchange for the group-based approach

This Go2 mode requires additional administration in advance defining groups-of-2 and supplying them with an additional attribute in addition to the group number mentioned for the other group-based modes - the subGroup attribute in the Association Mode record (see Section 3.2.2) of the PTP Key Request message. So, addressing for Go2 is achieved by use of the group number derived from domain number, sdoId and the additional attribute subGroup. Communication in that mode is performed using multicast addresses. If the latter

is undesirable, unicast addresses can be used but the particular IP or MAC addresses of the communication partners need to be configured upfront, too.

In spite of its specific name, Go2 allows more than two participants, for example additional Transparent Clocks. All participants in that subgroup need to be configured respectively. (To enable the NTS-KE server to supply the subgroup members with the particular security data the respective certificates may reflect permission to take part in the subgroup. Else another authorization method is to be used.)

Having predefined the Go2s the key management for this mode of operation follows the same procedure (see Figure 2) and uses the same NTS messages as the other group-based modes. Both participants, the Group-of-2 requester and the respective grantor need to have received their security parameters including key etc. before secure PTP communication can take place.

After the NTS key establishment messages for these group-based modes have been exchanged, the secured PTP communication can take place using the security association(s) communicated. The participants of the PTP network are now able to use the group key to verify secured PTP messages of the corresponding group or to generate secured PTP messages itself. In order to do this, the PTP node applies the group key together with the MAC algorithm to the PTP packet to generate the ICV transported in the AUTHENTICATION TLV of the PTP message.

The key management for these modes works relatively simple and needs only the above mentioned two NTS messages: PTP Key Request and PTP Key Response.

#### 2.1.2. NTS Message Exchange for the Ticket-based Approach

The scaling problems of the group-based approach are solved by the ticket-based approach (TiBA) for unicast connections. TiBA ensures end-to-end security between the two PTP communication partners, requester and grantor, and is therefore only suitable for PTP unicast where no group binding exists. Therefore, this model scales excellently with the number of connections. TiBA also allows free MAC algorithm and server negotiation, eliminating the need for the administrator to manually prepare the table of acceptable unicast masters at each individual PTP node. In addition, this allows optional load control by the NTS-KE server.

In (native) PTP unicast mode using unicast message negotiation ([IEEE1588-2019], Section 16.1) any potential instance (the grantor) which can be contacted by other PTP instances (the requesters) needs to register upfront with the NTS-KE server as depicted in Figure 3.

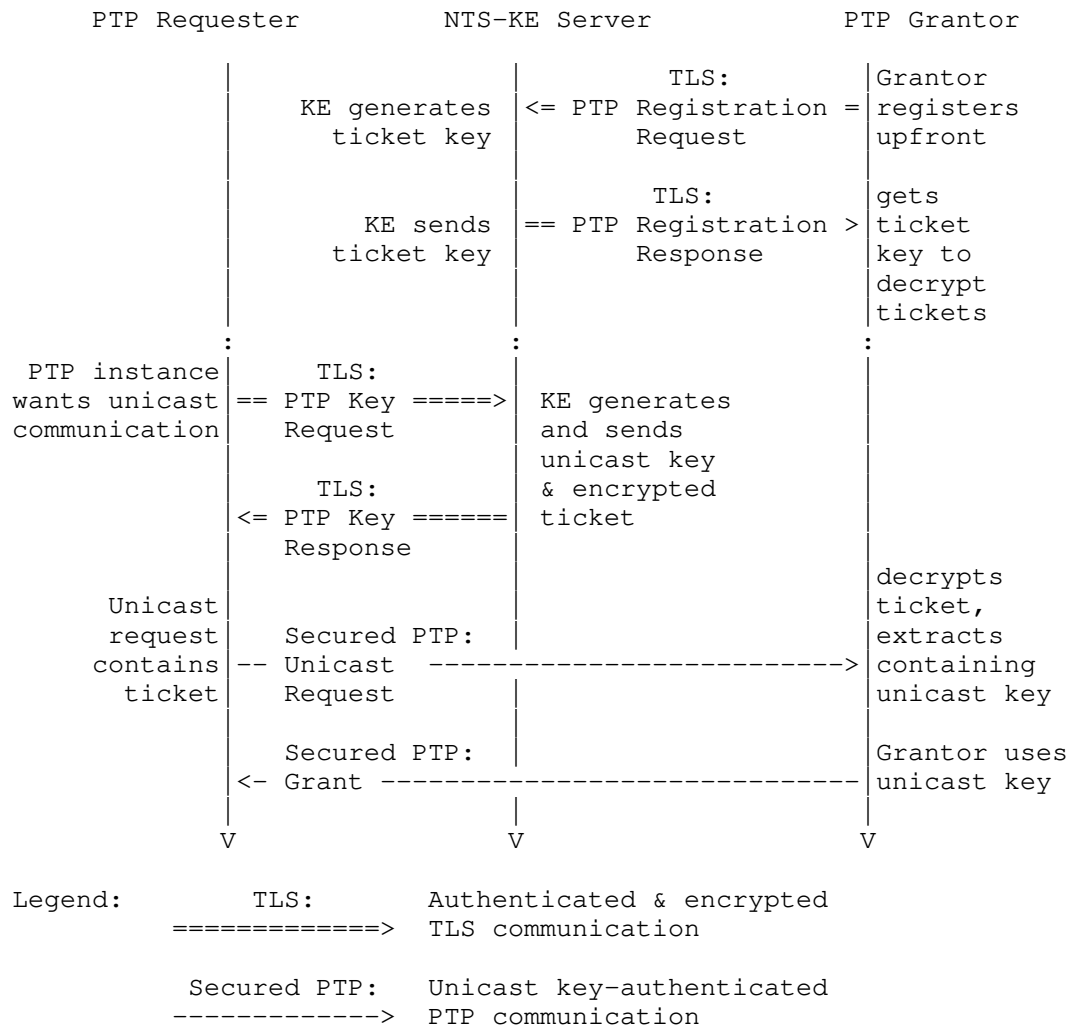


Figure 3: Message exchange for ticket-based unicast mode

(Note: As any PTP instance may request unicast messages from any other instance the terms requester and grantor as used in the standard suit better than talking about slave respectively master. In unicast PTP, the grantor is typically a PTP Port in the MASTER state, and the requester is typically a PTP Port in the SLAVE state. However all PTP Ports are allowed to grant and request unicast PTP message contracts regardless of which state they are in. A PTP port in MASTER state may be requester, a port in SLAVE state may be a grantor.)

Since the registration of unicast grantors is not provided for in the NTS-KE protocol, a new sub-protocol is needed, the NTS Time Server Registration (NTS-TSR) protocol. NTS-TSR does not conflict with NTS for NTP, and the original procedure for NTS-secured NTP remains unchanged. All NTS requests still arrive at the NTS-KE server on port 4460/TCP, whether a simple client or a time server connects. The authentication of the NTS-KE server by the querying partner already takes place when the TLS connection is established. In doing so, it chooses the NTS protocol to be used by selecting the ALPN [RFC7301]. If the ALPN contains the string "ntske/1", the NTS Key Establishment protocol is executed after the TLS handshake (see group-based approach). If it contains "ntstsr/1" instead, the NTS Time Server Registration protocol is executed. (Unlike the NTS-KE protocol, requesting grantors are already authenticated during the TLS handshake.)

The registration of a PTP grantor is performed via a PTP Registration Request message (see Section 2.3.3). The NTS-KE server answers with a PTP Registration Response message (see Section 2.3.4). If no delivery of security data is possible for whatever reason, the PTP Registration Response message contains a respective error code.

With the reception of the PTP Registration Response message, the grantor holds a ticket key known only to the NTS-KE server and the registered grantor. With this ticket key it can decrypt cryptographic information contained in a so-called ticket which enables secure unicast communication.

After the end of the registration process (phase 1), phase 2 begins with the key request of the client (now called requester). Similar to the group-based approach, a PTP instance (the requester) wanting to start a secured PTP unicast communication with a specific grantor contacts the NTS-KE server sending a PTP Key Request message (see Section 2.3.1) as shown in Figure 7, again using the TLS-secured NTS Key Establishment protocol. The NTS-KE server performs the authentication check of the client and then answers with a PTP Key Response message (see Section 2.3.2) with all the necessary data to begin the unicast communication with the desired partner or with a

respective error code if unicast communication with that instance is unavailable. Though the message types are the same as in GrBA the content differs.

The PTP Key Response message includes a unicast key to secure the PTP message exchange with the desired grantor. In addition, it contains the above mentioned (partially) encrypted ticket which the requester later (phase 3) transmits in a special Ticket TLV (see Section 5) with the secured PTP message to the grantor.

After the NTS key establishment messages for the PTP unicast mode have been exchanged, finally, the secured PTP communication (phase 3) can take place using the security association(s) communicated. A requester may send a (unicast key) secured PTP signaling message containing the received encrypted ticket, asking for a grant of a so-called unicast contract which contains a request for a specific PTP message type, as well as the desired frame rate.

The grantor receiving the PTP message decrypts the received ticket with its ticket key and extracts the containing security parameters, for example the unicast key used by the requester to secure the PTP message and the requester's identity. In that way the grantor can check the received message, identify the requester and can use the unicast key for further secure PTP communication with the requester until the unicast key expires.

A grantor that supports unicast and provides sufficient capacity will acknowledge the request for a unicast contract with a PTP unicast grant.

If a grantor is no longer at disposal for unicast mode during the lifetime of registration and ticket key, it sends a TLS-secured PTP Registration Revoke message (see Section 2.3.5, not shown in Figure 3) to the NTS-KE server, so requesters no longer receive PTP Key Response messages for this grantor.

The Heartbeat message (see Section 2.3.6, not shown in Figure 3) allows grantors to send messages to the NTS-KE server at regular intervals during the validity of the current security data and signal their own functionality. Optionally, these messages can contain status reports, for example, to enable load balancing between the registered time servers or to provide additional monitoring.

With its use of two protocols, the NTS-KE and the NTS-TSR protocol, this unicast mode is a bit more complex than the Group-of-2 approach and eventually uses all six new NTS messages. However, no subgroups have to be defined upfront. Addressing a grantor, the requesting instance simply may use the grantor's IP, MAC address or PortIdentity attribute.

## 2.2. General Topics

This section describes more general topics like key update and key generation as well as discussion of the time information on the NTS-KE server, the use of certificates and topics concerning upfront configuration.

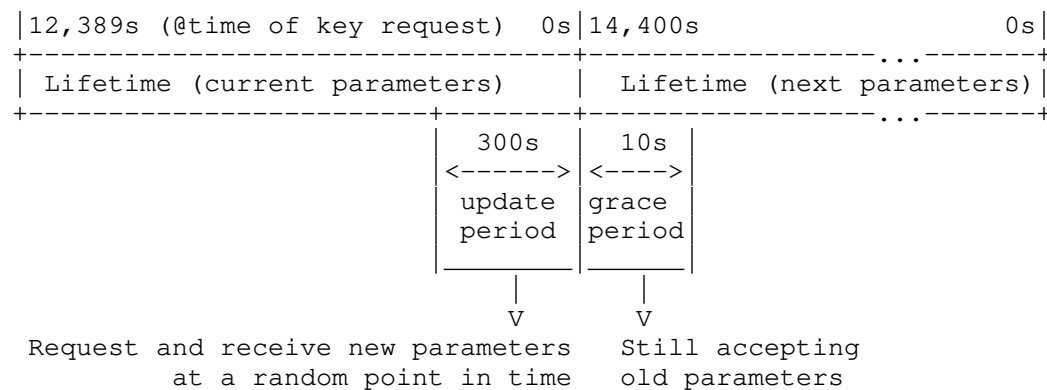
### 2.2.1. Key Update Process

The security parameters update process is an important part of NTS4PTP. It keeps the keys up to date, allows for both, runtime security policy changes and easy group control. The rotation operation allows uninterrupted PTP operation in multicast as well as unicast mode.

The update mechanism is based on the Validity Period record in the NTS response messages, which includes the three values lifetime, update period (UP) and grace period (GP), see Figure 4. The lifetime parameter specifies the validity period of the security parameters (e.g., security association (SA) and ticket) in seconds, which is counted down. This value can range from a few minutes to a few days. (Due to the design of the replay protection, a maximum lifetime of up to 388 days is possible, but should not be exploited). After the validity period has expired, the security parameters may no longer be used to secure PTP messages and must be deleted soon after.

New security parameters are available on the NTS-KE server during the update period, a time span before the expiry of the lifetime. The length of the update period is therefore always shorter than the full lifetime and is typically in the range of a few minutes. To ensure uninterrupted rotation for unicast connections, it is also necessary to ensure that the update period is greater than the minimum unicast contract time.

The grace period also helps to ensure uninterrupted key rotation. This value defines a period of time after the lifetime expiry during which the expired security parameters continue to be accepted. The grace period covers a few seconds at most and is only intended to compensate for runtime delays in the network during the update process. The respective values of the three parameters are defined by the administrator and can also be specified by a corresponding PTP profile.



Example:

```
-----
lifetime (full): 14,400s = 4h
update period:   300s = 5 min
grace period     10s
```

Figure 4: Example of the parameter rotation using lifetime, update period and grace period in group-based mode

As the value for lifetime is specified in seconds which denote the remaining time and is decremented down to zero, hard adjustments of the clock used have to be avoided. Therefore, the use of a monotonic clock is recommended. Requests during the currently running validity period will receive respectively adapted count values.

The Validity Period record (see Section 3.2.18) with its parameters lifetime, update time and grace period is contained in a so-called Current Parameters container record. Together with other security parameters this container record is always present in a PTP Key respectively Registration Response message. During the update period the response message additionally comprises the Next Parameters container record, which holds the new lifetime etc. starting at the end of the current lifetime as well as the other security parameters of the upcoming lifetime cycle.

Any PTP client sending a PTP Key Request to the NTS-KE server, be it in GrBA to receive the group SA or be it in TiBA asking for the unicast SA (unicast key etc. and encrypted ticket), will receive the Current Parameters container record where lifetime includes the remaining time to run rather than the full. Requesting during the update period the response includes also the new lifetime value in the Next Parameters container record. The new lifetime is the full value of the validity starting at the end of the current lifetime and update period. After the old lifetime has expired, only the new parameters (including lifetime, update period and grace period) have to be used. Merely during the grace period, the old SA will be accepted to cope with smaller delays in the PTP communication.

All PTP clients are obliged to connect to the NTS-KE server during the update period to allow for uninterrupted secured PTP operation. To avoid peak load on the NTS-KE server all clients SHOULD choose a random starting time during the update period.

In TiBA the unicast grantors execute the NTS-TSR protocol to register with the NTS-KE server. The rotation sequence (see Figure 5) and the behavior of the PTP Registration Response message is almost identical to the NTS-KE protocol. The main difference here is that the update period has to start earlier so that a grantor has re-registered before requesters ask for new security parameters at the NTS-KE server.

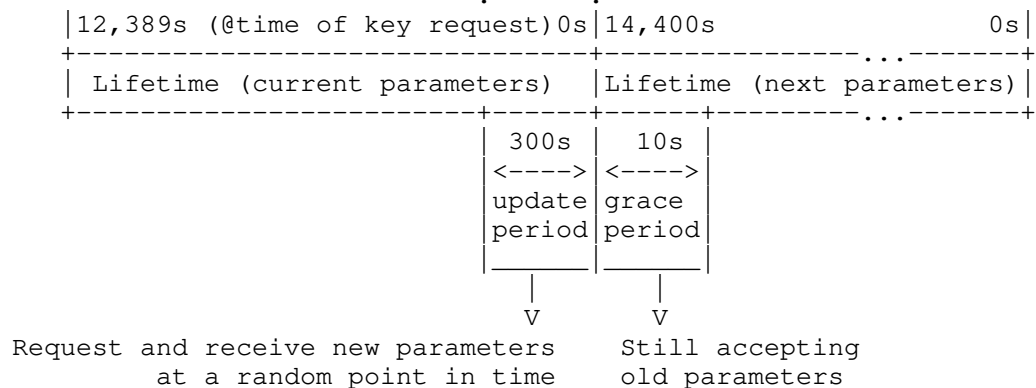
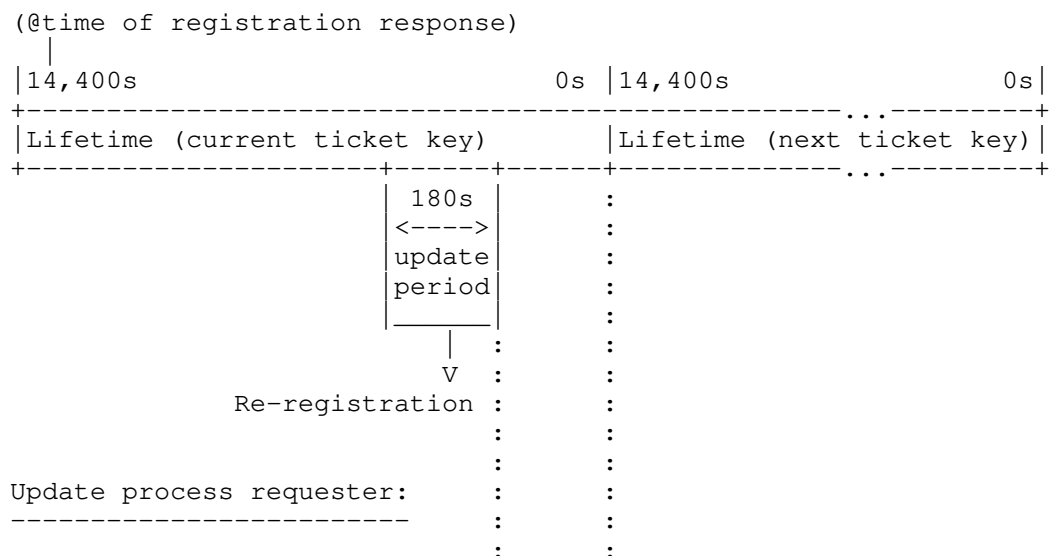
As the difference between the start of the requester's update period and the beginning of the update period of the grantor is not communicated, the grantor should contact the NTS-KE server directly after the start of its update period. However, since the rotation periods occur at different times for multiple grantors, no load peaks occur here either.

If a grantor does not re-register in time, requesters asking for a key etc. may not receive a Next Parameters container record, as no new SA is available at that point. So, requesters need to try again later in their update period.

As unicast contracts in TiBA run independently of the update cycle, a special situation may occur. If the remaining lifetime is short, it may be necessary to select a shorter time for the unicast contract validity period because the unicast contract cannot run longer than the lifetime. If a unicast contract is to be extended within the update period and the requester already owns the new ticket, it can already apply the upcoming security parameters here. This corresponds to some kind of negative grace period (pre-validity use of upcoming security parameters) and allows the requester to negotiate the full time for the unicast contract with the grantor.

If a grantor has revoked his registration with a PTP Registration Revoke message, requesters will receive a PTP Key Response message with an error code when trying to update for a new unicast key. No immediate key revoke mechanism exists. The grantor SHOULD not grant respective unicast requests during the remaining lifetime of the revoked key.

Update process grantor:



Example:

lifetime (full): 14,400s = 4h  
 update period grantor: 180s = 3 min  
 update period requester: 300s = 5 min  
 grace period: 10s

Figure 5: Example of the parameter rotation using lifetime and update period in ticket-based mode

### 2.2.2. Key Generation

In all cases keys obtained by a secure random number generator SHALL be used. The length of the keys depends on the MAC algorithm (see also last subsection in Section 4.2) respectively the AEAD algorithm utilized.

### 2.2.3. Time Information of the NTS-KE server

As the NTS-KE server embeds time duration information in the respective messages, its local time should be accurate to within a few seconds compared to the controlled PTP network(s). To avoid any dependencies, it should synchronize to a secure external time source, for example an NTS-secured NTP server. The time information is also necessary to check the lifetime of certificates used.

### 2.2.4. Certificates

The authentication of the TLS communication parties is based on certificates issued by a trusted Certificate Authority (CA) that are utilized during the TLS handshake. In classical TLS applications only servers are required to have them. For the key management system described here, the PTP nodes also need certificates to allow only authorized and trusted devices to get the group key and join a secure PTP network. (As TLS only authenticates the communication partners, authorization has to be managed by external means, see the topic "Authorization" in Section 2.2.5.4.) The verification of a certificate always requires a loose time synchronicity, because they have a validity period. This, however, reveals the well-known start-up problem, since secure time transfer itself requires valid certificates. (See the discussion and proposals on this topic in IETF RFC 8915 [RFC8915], Section 8.5 "Initial Verification of Server certificates" which applies to client and server certificates in the PTP key management system, too.)

Furthermore, some kind of Public Key Infrastructure (PKI) is necessary, which may be conceivable via the Online Certificate Status Protocol (OCSP) as well as offline via root CA certificates.

The TLS communication parties must be equipped with a private key and a certificate in advance. The certificate contains a digital signature of the CA as well as the public key of the sender. The key pair is required to establish an authenticated and encrypted channel for the initial TLS phase. Distribution and update of the certificates can be done manually or automatically. However, it is important that they are issued by a trusted CA instance, which can be either local (private CA) or external (public CA).

For the certificates the standard for X.509 [ITU-T\_X.509] certificates MUST be used. Additional data in the certificates like domain, sdoId and/or subgroup attributes may help in authorizing. In that case it should be noted that using the PTP device in another network then implies to have a new certificate, too. Working with certificates without authorization information would not have that disadvantage, but more configuring at the NTS-KE server would be necessary: which domain, sdoId and/or subgroup attributes belong to which certificate.

As TLS is used to secure both sub protocols, the NTS KE and the NTS-TSR protocol, a comment on the security of TLS seems reasonable. A TLS 1.3 connection is considered secure today. However, note that a DoS (Denial of Service) attack on the key server can prevent new connections or parameter updates for secure PTP communication. A hijacked key management system is also critical, because it can completely disable the protection mechanism. A redundant implementation of the key server is therefore essential for a robust system. A further mitigation can be the limitation of the number of TLS requests of single PTP nodes to prevent flooding. But such measures are out of the scope of this document.

#### 2.2.5. Upfront Configuration

All PTP instances as well as the NTS-KE server need to be configured by the network administrator. This applies to several fields of parameters.

##### 2.2.5.1. Security Parameters

The cryptographic algorithm and associated parameters (the so-called security association(s) - SA) used for PTP keys are configured by network operators at the NTS-KE server. PTP instances that do not support the configured algorithms cannot operate with the security. Since most PTP networks are managed by a single organization, configuring the cryptographic algorithm (MAC) for ICV calculation is practical. This prevents the need for the NTS-KE server and PTP instances to implement an NTS algorithm negotiation protocol.

For the ticket-based approach the AEAD algorithms need to be specified which the PTP grantors and the NTS-KE server support and negotiate during the registration process. Optionally, the MAC algorithm may be negotiated during a unicast PTP Key Request to allow faster or stronger algorithms, but a standard protocol supported by every instance should be defined. Eventually, suitable algorithms may be defined in a respective PTP profile.

#### 2.2.5.2. Key Lifetimes

Supplementary to the above mentioned SAs the desired key rotation periods, i.e., the lifetimes of keys respectively all security parameters need to be configured at the NTS-KE server. This applies to the lifetime of a group key in the group-based approach as well as the lifetime of ticket key and unicast key in the ticket-based unicast approach (typically for every unicast pair in general or eventually specific for each requestor-grantor pair). In addition, the corresponding update periods and grace periods need to be defined. Any particular lifetime, update period and grace period is configured as time spans specified in seconds.

#### 2.2.5.3. Certificates

The network administrator has to supply each PTP instance and the NTS-KE server with their X.509 certificates. The TLS communication parties must be equipped with a private key and a certificate containing the public key in advance (see Section 2.2.4).

#### 2.2.5.4. Authorization

The certificates provide authentication of the communication partners. Normally, they do not contain authorization information. Authorization decides, which PTP instances are allowed to join a group (in any of the group-based modes) or may enter a unicast communication in the ticket-based approach and request the respective SA(s) and key.

As mentioned, members of a group (multicast mode, mixed multicast/unicast mode) are identified by their domain and their sdoId. PTP domain and sdoId may be attributes in the certificates of the potential group members supplying additional authorization. If not contained in the certificates extra authorization means are necessary. (See also the discussion on advantages and disadvantages on certificates containing additional authorization data in Section 2.2.4.)

If the special Group-of-2 mode is used, the optional subGroup parameter (i.e., the subgroup number) needs to be specified at all members of respective Go2s, upfront. To enable the NTS-KE server to supply the subgroup members with the particular security data their respective certificates may reflect permission to take part in the subgroup. Else another authorization method is to be used.

In native unicast mode, any authenticated grantor that is member of the group used for multicast may request a registration for unicast communication at the NTS-KE server. If it is intended for unicast,

this must be configured locally. If no group authorization is available (e.g., pure unicast operation) another authentication scheme is necessary.

In the same way, any requester (if configured for it locally) may request security data for a unicast connection with a specific grantor. Only authentication at the NTS-KE server using its certificate and membership in the group used for multicast is needed. If a unicast communication is not desired by the grantor, it should not grant a specific unicast request. Again, if no group authorization is available (e.g., pure unicast operation) another authentication scheme is necessary.

Authorization can be executed at least in some manual configuration. Probably the application of a standard access control system like Diameter, RADIUS or similar would be more appropriate. Also role-based access control (RBAC), attribute-based access control (ABAC) or more flexible tools like Open Policy Agent (OPA) could help administering larger systems. But details of the authorization of PTP instances lies out of scope of this document.

#### 2.2.5.5. Transparent Clocks

Transparent Clocks (TC) need to be supplied with respective certificates, too. For group-based modes they must be configured for the particular PTP domain and sdoId and eventually for the specific subgroup(s) when using Group-of-2. They need to request for the relevant group key(s) at the NTS-KE server to allow secure use of the correctionfield in a PTP message and generation of a corrected ICV. If TCs are used in ticket-based unicast mode, they need to be authorized for the particular unicast path.

Authorization of TCs for the respective groups, subgroups and unicast connections is paramount. Otherwise the security can easily be broken with attackers pretending to be TCs in the path. Authorization of TCs is necessary too in unicast communication, even if the normal unicast partners need not be especially authorized.

Transparent clocks may notice that the communication runs secured. In the group-based approaches multicast mode and mixed multicast/unicast mode they construct the GroupID from domain and sdoId and request a group key from the NTS-KE server. Similarly, they can use the additional subgroup attribute in Go2 mode for a (group) key request. Afterwards they can check the ICV of incoming messages, fill in the correction field and generate a new ICV for outgoing messages. In ticket-based unicast mode a TC may notice a secured unicast request from a requester to the grantor and can request the unicast key from the NTS-KE server to make use of the correction

field afterwards. As mentioned above upfront authentication and authorization of the particular TCs is paramount not to open the secured communication to attackers.

#### 2.2.5.6. Start-up considerations

At start-up of a single PTP instance or the complete PTP network some issues have to be considered.

At least loose time synchronization is necessary to allow for authentication using the certificates. See the discussion and proposals on this topic in IETF RFC 8915 [RFC8915], Section 8.5 "Initial Verification of Server certificates" which applies to client certificates in the PTP key management system, too.

Similarly, to a key re-request during an update period, key requests SHOULD be started at a random point in time after start-up to avoid peak load on the NTS-KE server. Every grantor must register with the NTS-KE server before requesters can request a unicast key (and ticket).

### 2.3. Overview of NTS Messages and their Structure for Use with PTP

Section 2.1 described the principle communication sequences for PTP Key Request, PTP Registration Request and corresponding response messages. All messages follow the "NTS Key Establishment Process" stated in the first part (until the description of Figure 3 starts) of Section 4 of IETF RFC 8915 [RFC8915]:

\_The NTS key establishment protocol is conducted via TCP port 4460. The two endpoints carry out a TLS handshake in conformance with Section 3, with the client offering (via an ALPN extension [RFC7301])\_, and the server accepting, an application-layer protocol of "ntske/1". Immediately following a successful handshake, the client SHALL send a single request as Application Data encapsulated in the TLS-protected channel. Then, the server SHALL send a single response. After sending their respective request and response, the client and server SHALL send TLS "close\_notify" alerts in accordance with Section 6.1 of RFC 8446 [RFC8446].

\_The client's request and the server's response each SHALL consist of a sequence of records formatted according to\_ Figure 6\_. The request and a non-error response each SHALL include exactly one NTS Next Protocol Negotiation record. The sequence SHALL be terminated by a "End of Message" record. The requirement that all NTS-KE messages be terminated by an End of Message record makes them self-delimiting.\_

\_Clients and servers MAY enforce length limits on requests and responses, however, servers MUST accept requests of at least 1024 octets and clients SHOULD accept responses of at least 65536 octets.\_

\_The fields of an NTS-KE record are defined as follows:\_

- \_C (Critical Bit): Determines the disposition of unrecognized Record Types. Implementations which receive a record with an unrecognized Record Type MUST ignore the record if the Critical Bit is 0 and MUST treat it as an error if the Critical Bit is 1 (see Section 4.1.3).\_
- \_Record Type Number: A 15-bit integer in network byte order. The semantics of record types 0-7 are specified in this memo. Additional type numbers SHALL be tracked through the IANA Network Time Security Key Establishment Record Types registry.\_
- \_Body Length: The length of the Record Body field, in octets, as a 16-bit integer in network byte order. Record bodies MAY have any representable length and need not be aligned to a word boundary.\_
- \_Record Body: The syntax and semantics of this field SHALL be determined by the Record Type.\_

\_For clarity regarding bit-endianness: the Critical Bit is the most-significant bit of the first octet. In the C programming language, given a network buffer 'unsigned char b[]' containing an NTS-KE record, the critical bit is 'b[0] >> 7' while the record type is '((b[0] & 0x7f) << 8) + b[1]'.\_"\_

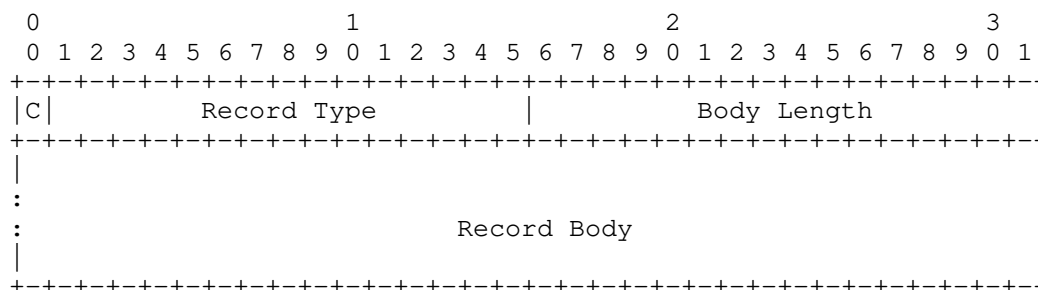


Figure 6: NTS-KE record format

Thus, all NTS messages consist of a sequence of records, each containing a Critical Bit C, the Record Type, the Body Length and the Record Body, see Figure 6. More details on record structure as well as the specific records used here are given in Section 3 and respective subsections there. So-called container records (short: container) themselves comprise a set of records in the record body that serve a specific purpose, e.g., the Current Parameters container record.

The records contained in a message may follow in arbitrary sequence (though nothing speaks against using the sequence given in the record descriptions), only the End of Message record has to be the last one in the sequence indicating the end of the current message. Container records do not include an End of Message record.

The NTS key management for PTP is based on six new NTS messages:

- \* PTP Key Request message (see Section 2.3.1)
- \* PTP Key Response message (see Section 2.3.2)
- \* PTP Registration Request message (see Section 2.3.3)
- \* PTP Registration Response message (see Section 2.3.4)
- \* PTP Registration Revoke message (see Section 2.3.5)
- \* Heartbeat message (see Section 2.3.6)

The following sections describe the principle structure of those new NTS messages for the PTP key management. More details especially on the records the messages are built of and their types, sizes, requirements and restrictions are given in Section 3.2.

### 2.3.1. PTP Key Request Message

PTP Key Request (NTS-KE protocol)

Record	Exemplary body contents
NTS Next Protocol Negotiation	PTPv2.1
Association Mode	{Assoc.Type  Assoc.Val.}
Supported MAC Algorithms (opt.)	CMAC
Source PortIdentity (unicast only)	{binary data}
End of Message	

Figure 7: Structure of a PTP Key Request message

Figure 7 shows the record structure of a PTP Key Request message. In the right column typical values are shown as examples. Detailed information on types, sizes etc. is given in Section 3.2. The message starts with the NTS Next Protocol Negotiation record which in this application always holds PTPv2.1. The following Association Mode record describes the mode how the PTP instance wants to communicate: In the group-based approach the desired group number

(plus eventually the subgroup attribute) is given. For ticket-based unicast communication the Association Mode contains the identification of the desired grantor, for example IPv4 and its IP address.

Only in TiBA, an optional record may follow. It offers the possibility to choose from additional MAC algorithms and presents the supported algorithms from which the NTS-KE server may choose. Again, only in ticket-based unicast mode, the Source PortIdentity record gives the data of the identification of the applying requester, for example IPv4 and its IP address. The messages always end with an End of Message record.

### 2.3.2. PTP Key Response Message

Figure 8 shows the record structure of a PTP Key Response message from the NTS-KE server (NTS-KE protocol). In the right column typical values are shown as examples. Detailed information on types, sizes etc. is given in Section 3.2. The message starts with the NTS Next Protocol Negotiation record which in this application always holds PTPv2.1.

PTP Key Response (NTS-KE protocol)

Record	Exemplary body contents
NTS Next Protocol Negotiation	PTPv2.1
Current Parameters	set of Records {...}
Next Parameters	set of Records {...}
End of Message	

PTP Key Response (NTS-KE protocol) - in case of an error

Record	Exemplary body contents
NTS Next Protocol Negotiation	PTPv2.1
Error	Not authorized
End of Message	

Figure 8: Structure of a PTP Key Response message.

The following Current Parameters record is a container record containing in separate records all the security data needed to join and communicate in the secured PTP communication during the current validity period. Figure 9 gives an example of data contained in that record. For more details on the records contained in the Current Parameters container record see Section 3.2.3.

Current Parameters container record (PTP Key Response)

Record	Exemplary body contents
Security Association	data set {...}
Validity Period	{1560s    300s    10s}
PTP Time Server (unicast only)	data set {...}
Ticket (unicast only)	data set {...}

Figure 9: Exemplary contents of a Current Parameters container record of a PTP Key Response message

If the request lies inside the update period, a Next Parameters container record is additionally appended in the PTP Key Response message giving all the security data needed in the upcoming validity period. Its structure follows the same composition as the Current Parameters container record. In case of an error, both parameters container records are removed and a single error record is inserted (see the lower part of Figure 8). The messages always end with an End of Message record.

### 2.3.3. PTP Registration Request Message

PTP Registration Request (NTS-TSR protocol)

Record	Exemplary body contents
NTS Message Type	PTP Registration Request    v1.0
PTP Time Server	data set {...}
AEAD Algorithm Negotiation	{AEAD_512    AEAD_256}
Supported MAC Algorithms	{CMAC    HMAC}
End of Message	

Figure 10: Structure of a PTP Registration Request message

The PTP Registration Request message (NTS-TSR protocol) starts with the NTS Message Type record containing the message type as well as the message version number, here always 1.0, see Figure 10. (As the message belongs to the NTS-TSR protocol, no NTS Next Protocol Negotiation record is necessary.)

The PTP Time Server record presents all known network addresses of this grantor that are supported for a unicast connection. The following AEAD Algorithm Negotiation record indicates which algorithms for encryption of the ticket the grantor supports.

Then the next record (not optional as in PTP Key Request) follows, presenting all the grantor's supported MAC algorithms. The Supported MAC Algorithms record contains a list and comprises the MAC algorithms supported by the grantor that are feasible for calculating the ICV when securing the PTP messages in TiBA. The message always ends with an End of Message record.

#### 2.3.4. PTP Registration Response Message

## PTP Registration Response (NTS-TSR protocol)

Record	Exemplary body contents
NTS Message Type	PTP Registration Response    v1.0
Current Parameters	set of Records {...}
Next Parameters	set of Records {...}
Heartbeat Timeout (opt.)	900s
End of Message	

## PTP Registration Response (NTS-TSR protocol)- in case of an error

Record	Exemplary body contents
NTS Message Type	PTP Registration Response    v1.0
Error	Not authorized
End of Message	

Figure 11: Structure of a PTP Registration Response message

The PTP Registration Response message (NTS-TSR protocol) from the NTS-KE server starts with the NTS Message Type record containing the message type as well as the message version number, here always 1.0, see Figure 11. (As the message belongs to the NTS-TSR protocol, no NTS Next Protocol Negotiation record is necessary.)

As in the NTS-KE protocol, the following Current Parameters record is a container record containing in separate records all the necessary parameters for the current validity period. Figure 12 gives an example of data contained in that record. For more details on the records contained in the Current Parameters container record see Section 3.2.3.

Current Parameters container record (PTP Registration Response)	
Record	Exemplary body contents
AEAD Algorithm Negotiation	AEAD_AES_SIV_CMAC_512
Validity Period	{2460s    400s    10s}
Ticket Key ID	278
Ticket Key	{binary data}

Figure 12: Exemplary contents of a Current Parameters container record of a PTP Registration Response message in the NTS-TSR protocol

If the registration request lies inside the update period a Next Parameters container record is additionally appended giving all the security data needed in the upcoming validity period. Its structure follows the same composition as the Current Parameters container record. The messages always end with an End of Message record.

#### 2.3.5. PTP Registration Revoke Message

PTP Registration Revoke (NTS-TSR protocol)	
Record	Exemplary body contents
NTS Message Type	PTP Registr. Revoke    v1.0
Source PortIdentity	{binary data}
End of Message	

Figure 13: Structure of a PTP Registration Revoke message

The PTP Registration Revoke message (NTS-TSR protocol) from the grantor starts with the NTS Message Type record containing the message type as well as the message version number, here always 1.0, see Figure 13. (As the message belongs to the NTS-TSR protocol, no NTS Next Protocol Negotiation record is necessary.)

The second record contains the Source PortIdentity which identifies the grantor wanting to stop its unicast support. This allows the NTS-KE server to uniquely identify the grantor if the PTP device communicates with the NTS-KE server via a management port running multiple grantors. The message always ends with an End of Message record.

#### 2.3.6. Heartbeat Message

Heartbeat (NTS-TSR protocol)

Record	Exemplary body contents
NTS Message Type	Heartbeat    v1.0
Status (optional)	server load: low
End of Message	

Figure 14: Structure of a Heartbeat message in the NTS-TSR protocol

The Heartbeat message (NTS-TSR protocol) from the grantor to the NTS-KE server starts with the NTS Message Type record containing the message type as well as the message version number, here always 1.0, see Figure 14. (As the message belongs to the NTS-TSR protocol, no NTS Next Protocol Negotiation record is necessary.)

The second record contains the optional Status record which allows the grantor to present various status updates to the NTS-KE server. The message always ends with an End of Message record.

Heartbeat messages provide grantors with the ability to send messages to the NTS-KE server at regular intervals to signal their own functionality. These messages can optionally also contain one or multiple status records (see Figure 14), for example to improve load balancing between the registered time servers or to provide additional monitoring. The NTS-KE server **MUST** accept Heartbeat messages from a grantor if they have been previously requested by the NTS-KE server in the Registration Response message. However, the NTS-KE server **MAY** discard heartbeat messages if they arrive more frequently than specified by the heartbeat timeout (see Section 2.3.6). If the NTS-KE server receives heartbeat messages from a grantor even though it is not requested, the NTS-KE server **SHOULD** discard these messages and not process them further. Processing of the status information is optional and the status records **MAY** be ignored by the NTS-KE server. If the Grantor sends heartbeat messages to the NTS-KE server, the frames **SHOULD NOT** exceed the maximum transmission unit (MTU, 1500 octets for Ethernet).

### 3. NTS Messages for PTP

This section covers the structure of the NTS messages and the details of the respective payload. The individual parameters are transmitted by NTS records, which are described in more detail in Section 3.2. In addition to the NTS records defined for NTP in IETF RFC8915, further records are required, which are listed in Table 1 below and begin with Record Type 1024 (compare IETF RFC 8915 [RFC8915], Section 7.6. Network Time Security Key Establishment Record Types Registry).

NTS Record Types	Description	Record Used in Protocol	Reference
0	End of Message	NTS-KE/ NTS-TSR	[RFC8915], Section 4.1.1; this document, Section 3.2.4
1	NTS Next Protocol Negotiation	NTS-KE	[RFC8915], Section 4.1.2; this document, Section 3.2.8
2	Error	NTS-KE/ NTS-TSR	[RFC8915], Section 4.1.3; this document, Section 3.2.5
3	Warning	NTS-KE	[RFC8915], Section 4.1.4; not used

			for PTP
4	AEAD Algorithm Negotiation	NTS-TSR	[RFC8915], Section 4.1.5; this document, Section 3.2.1
5	New Cookie for NTPv4	NTS-KE	[RFC8915], Section 4.1.6; not used for PTP
6	NTPv4 Server Negotiation	NTS-KE	[RFC8915], Section 4.1.7; not used for PTP
7	NTPv4 Port Negotiation	NTS-KE	[RFC8915], Section 4.1.8; not used for PTP
8 – 1023	Reserved for NTP		
1024	Association Mode	NTS-KE	This document, Section 3.2.2
1025	Current Parameters	NTS-KE/ NTS-TSR	This document, Section 3.2.3
1026	Heartbeat Timeout	NTS-TSR	This document, Section 3.2.6
1027	Next Parameters Container	NTS-KE/ NTS-TSR	This document, Section 3.2.7
1028	NTS Message Type	NTS-TSR	This document, Section 3.2.9
1029	PTP Time Server	NTS-KE/ NTS-TSR	This document, Section 3.2.10
1030	Security Association	NTS-KE	This document, Section 3.2.11
1031	Source PortIdentity	NTS-KE/ NTS-TSR	This document, Section 3.2.12
1032	Status	NTS-TSR	This document, Section 3.2.13

1033	Supported MAC Algorithms	NTS-KE/ NTS-TSR	This document, Section 3.2.14
1034	Ticket	NTS-TSR	This document, Section 3.2.15
1035	Ticket Key	NTS-TSR	This document, Section 3.2.16
1036	Ticket Key ID	NTS-TSR	This document, Section 3.2.17
1037	Validity Period	NTS-KE/ NTS-TSR	This document, Section 3.2.18
1038 - 16383	Unassigned		
16384 - 32767	Reserved for Private or Experimental Use		[RFC8915]

Table 1: NTS Key Establishment and Time Server Registration  
record types registry

### 3.1. NTS Message Types

This section repeats the composition of the specific NTS messages for the PTP key management in overview form. The specification of the respective records from which the messages are constructed follows in Section 3.2. The reference column in the tables refer to the specific subsections.

The NTS messages MUST contain the records given for the particular message though not necessarily in the same sequence indicated. Only the End of Message record MUST be the final record.

\*PTP Key Request (NTS-KE protocol)\*

NTS Record Name	Mode*	Use	Reference
NTS Next Protocol Negotiation	GrBA / TiBA	mandatory	This document, Section 3.2.8

Association Mode	GrBA / TiBA	mandatory	This document, Section 3.2.2
Supported MAC Algorithms	TiBA	optional	This document, Section 3.2.14
Source PortIdentity	TiBA	mandatory	This document, Section 3.2.12
End of Message	GrBA / TiBA	mandatory	This document, Section 3.2.4

Table 2: Record structure of the PTP Key Request message

\* The Mode column refers to the intended use of the particular record for the respective PTP communication mode.

\*PTP Key Response (NTS-KE protocol)\*

NTS Record Name	Mode	Use	Reference
NTS Next Protocol Negotiation	GrBA / TiBA	mandatory	This document, Section 3.2.8
Current Parameters	GrBA / TiBA	mandatory	This document, Section 3.2.3
Next Parameters Container	GrBA / TiBA	mandatory (only during update period)	This document, Section 3.2.7
End of Message	GrBA / TiBA	mandatory	This document, Section 3.2.4

Table 3: Record structure of the PTP Key Response message.  
In case of an error, both parameters container records are removed and a single error record is inserted.

The structure of the respective container records (Current Parameters and Next Parameters) used in the PTP Key Response message is given below:

\*Current/Next Parameters container - PTP Key Response (NTS-KE protocol)\*

NTS Record Name	Mode	Use	Reference
Security Association	GrBA / TiBA	mandatory	This document, Section 3.2.11
Validity Period	GrBA / TiBA	mandatory	This document, Section 3.2.18
PTP Time Server	TiBA	mandatory	This document, Section 3.2.10
Ticket	TiBA	mandatory	This document, Section 3.2.15

Table 4: Record structure of the container records

\*PTP Registration Request (NTS-TSR protocol)\*

NTS Record Name	Mode	Use	Reference
NTS Message Type	TiBA	mandatory	This document, Section 3.2.9
PTP Time Server	TiBA	mandatory	This document, Section 3.2.10
AEAD Algorithm Negotiation	TiBA	mandatory	This document, Section 3.2.1
Supported MAC Algorithms	TiBA	mandatory	This document, Section 3.2.14
End of Message	TiBA	mandatory	This document, Section 3.2.4

Table 5: Record structure of the PTP Registration Request message

\*PTP Registration Response (NTS-TSR protocol)\*

NTS Record Name	Mode	Use	Reference
NTS Message Type	TiBA	mandatory	This document,

			Section 3.2.9
Current Parameters	TiBA	mandatory	This document, Section 3.2.3
Next Parameters	TiBA	mandatory (only during update period)	This document, Section 3.2.7
Heartbeat Timeout	TiBA	optional	This document, Section 3.2.6
End of Message	TiBA	mandatory	This document, Section 3.2.4

Table 6: Record structure of the PTP Registration Response message. In case of an error, both parameters container records are removed and a single error record is inserted.

The structure of the respective container records (Current Parameters and Next Parameters ) used in the PTP Registration Response message is given below:

\*Current/Next Parameters container - PTP Registration Response (NTS-TSR protocol)\*

NTS Record Name	Mode	Use	Reference
AEAD Algorithm Negotiation	TiBA	mandatory	This document, Section 3.2.1
Validity Period	TiBA	mandatory	This document, Section 3.2.18
Ticket Key ID	TiBA	mandatory	This document, Section 3.2.17
Ticket Key	TiBA	mandatory	This document, Section 3.2.16

Table 7: Record structure of the container records in the PTP Registration Response message

\*PTP Registration Revoke (NTS-TSR protocol)\*

=====

NTS Record Name	Mode	Use	Reference
NTS Message Type	TiBA	mandatory	This document, Section 3.2.9
Source PortIdentity	TiBA	mandatory	This document, Section 3.2.12
End of Message	TiBA	mandatory	This document, Section 3.2.4

Table 8: Record structure of the PTP Registration Revoke message

\*Heartbeat Message (NTS-TSR protocol)\*

NTS Record Name	Mode	Use	Reference
NTS Message Type	TiBA	mandatory	This document, Section 3.2.9
Status	TiBA	optional	This document, Section 3.2.13
End of Message	TiBA	mandatory	This document, Section 3.2.4

Table 9: Record structure of the Heartbeat message in the NTS-TSR protocol

### 3.2. NTS Records

The following subsections describe the specific NTS records used to construct the NTS messages for the PTP key management system in detail. They appear in alphabetic sequence of their individual names. See Section 3.1 for the application of the records in the respective messages.

Note: For easier editing of the content, most of the descriptions in the following subsections are written as bullet points.

#### 3.2.1. AEAD Algorithm Negotiation

Used in NTS-TSR protocol

This record is required in unicast mode and enables the negotiation of the AEAD algorithm needed to encrypt and decrypt the ticket. The negotiation takes place between the PTP grantor and the NTS-KE server by using the NTS registration messages. The structure and properties follow the record defined in IETF RFC 8915 [RFC8915], Section 4.1.5.

Content and conditions:

- \* The record has a Record Type number of 4 and the Critical Bit MAY be set.
- \* The Record Body contains a sequence of 16-bit unsigned integers in network byte order:  
 \*Supported AEAD Algorithms = {AEAD 1 || AEAD 2 || ...}\*
- \* Each integer represents a numeric identifier of an AEAD algorithm registered by the IANA. (<https://www.iana.org/assignments/aead-parameters/aead-parameters.xhtml>)
- \* Duplicate identifiers SHOULD NOT be included.
- \* Grantor and NTS-KE server MUST support at least the AEAD\_AES\_SIV\_CMAC\_256 algorithm.
- \* A list of recommended AEAD algorithms is shown in the following Table 10.
- \* Other AEAD algorithms MAY also be used.

Numeric ID	AEAD Algorithm	Use	Key Length (Octets)	Reference
15	AEAD_AES_SIV_CMAC_256	mand.	16	[RFC5297]
16	AEAD_AES_SIV_CMAC_384	opt.	24	[RFC5297]
17	AEAD_AES_SIV_CMAC_512	opt.	32	[RFC5297]
32 - 32767	Unassigned			
32768 - 65535	Reserved for Private or Experimental Use			[RFC5116]

Table 10: AEAD algorithms

- \* In a PTP Registration Request message, this record MUST be contained exactly once.
- \* In that message at least the AEAD\_AES\_SIV\_CMAC\_256 algorithm MUST be included.

- \* If multiple AEAD algorithms are supported, the grantor SHOULD put the algorithm identifiers in descending priority in the Record Body.
- \* Strong algorithms with higher bit lengths SHOULD have higher priority.
- \* In a PTP Registration Response message, this record MUST be contained exactly once in the Current Parameters container record and exactly once in the Next Parameters container record.
- \* The Next Parameters container record MUST be present only during the update period.
- \* The NTS-KE server SHOULD choose the highest priority AEAD algorithm from the request message that grantor and NTS-KE server support.
- \* The NTS-KE server MAY ignore the priority and choose a different algorithm that grantor and NTS-KE server support.
- \* In a PTP Registration Response message, this record MUST contain exactly one AEAD algorithm.
- \* The selected algorithm MAY differ in the corresponding Current Parameters container record and Next Parameters container record.

### 3.2.2. Association Mode

Used in NTS-KE protocol

This record enables the NTS-KE server to distinguish between a group based request (multicast, mixed multicast/unicast, Group-of-2) or a unicast request. A multicast request carries a group number, while a unicast request contains an identification attribute of the grantor (e.g., IP address or PortIdentity).

Content and conditions:

- \* In a PTP Key Request message, this record MUST be contained exactly once.
- \* The record has a Record Type number of 1024 and the Critical Bit MAY be set.
- \* The Record Body SHALL consist of two data fields:

field	Octets	Offset
Association Type	2	0
Association Value	A	2

Table 11: Association

- \* The Association Type is a 16-bit unsigned integer.
- \* The length of Association Value depends on the value of Association Type.
- \* All data in the fields are stored in network byte order.
- \* The type numbers of Association Type as well as the length and content of Association Value are shown in the following table and more details are given below.

Description	Assoc. Type Number	Association Mode	Association Value Content	Assoc. Value Octets
Group	0	Multicast / Unicast*	Group Number	5
IPv4	1	Unicast	IPv4 address of the target port	4
IPv6	2	Unicast	IPv6 address of the target port	16
802.3	3	Unicast	MAC address of the target port	6
PortIdentity	4	Unicast	PortIdentity of the target PTP entity	10

Table 12: Association Types

Unicast\*: predefined groups of two (Group-of-2, Go2, see Group entry below)

Group:

- \* This association type allows a PTP instance to join a PTP multicast group.
- \* A group is identified by the PTP domain, the PTP profile (sdoId) and a sub-group attribute (see table below).
- \* The PTP domainNumber is an 8-bit unsigned integer in the closed range 0 to 255.
- \* The sdoId of a PTP domain is a 12-bit unsigned integer in the closed range 0 to 4095:

- The most significant 4 bits are named the majorSdoId.
- The least significant 8 bits are named the minorSdoId.
- Reference: IEEE Std 1588-2019, Section 7.1.1

\*sdoId = {majorSdoId || minorSdoId}\*

- \* The subGroup is 16-bit unsigned integer, which allows the division of a PTP multicast network into separate groups, each with individual security parameters.
- \* This also allows manually configured unicast connections (Group-of-2), which can include transparent clocks as well.
- \* The subGroup number is defined manually by the administrator.
- \* Access to the groups is controlled by authorization procedures of the PTP devices (see Section 2.2.5.4).
- \* If no subgroups are required (= multicast mode), this attribute MUST contain the value zero.
- \* The group number is eventually formed by concatenation of the following values:  
\*group number = {domainNumber || 4 bit zero padding || sdoId || subGroup}\*

This is equivalent to:

Bits 7 - 4	Bits 3 - 0	Octets	Offset
domainNumber (high)	domainNumber (low)	1	0
zero padding	majorSdoId	1	1
minorSdoId (high)	minorSdoId (low)	1	2
subgroup (high)	subGroup (low)	2	4

Table 13: Group Association

IPv4:

- \* This Association Type allows a requester to establish a PTP unicast connection to the desired grantor.
- \* The Association Value contains the IPv4 address of the target PTP entity.
- \* The total length is 4 octets.

IPv6:

- \* This Association Type allows a requester to establish a PTP unicast connection to the desired grantor.

- \* The Association Value contains the IPv6 address of the target PTP entity.
- \* The total length is 16 octets.

#### 802.3:

- \* This Association Type allows a requester to establish a PTP unicast connection to the desired grantor.
- \* The Association Value contains the MAC address of the Ethernet port of the target PTP entity.
- \* The total length is 6 octets.
- \* This method supports the 802.3 mode in PTP, where no UDP/IP stack is used.

#### PortIdentity:

- \* This Association Type allows a requester to establish a PTP unicast connection to the desired grantor.
- \* The Association Value contains the PortIdentity of the target PTP entity.
- \* The total length is 10 octets.
- \* The PortIdentity consists of the attributes clockIdentity and portNumber:  
\*PortIdentity = {clockIdentity || portNumber}\*
- \* The clockIdentity is an 8 octet array and the portNumber is a 16-bit unsigned integer.
- \* Source: IEEE Std 1588-2019, Sections 5.3.5 and 7.5

### 3.2.3. Current Parameters

Used in NTS-KE and NTS-TSR protocol

This record is a simple container that can carry an arbitrary number of NTS records. It holds all security parameters relevant for the current validity period. The content as well as further conditions are defined by the respective NTS messages. The order of the included records is arbitrary and the parsing rules are so far identical with the NTS message. One exception: An End of Message record SHOULD NOT be present and MUST be ignored. When the parser reaches the end of the Record Body quantified by the Body Length, all embedded records have been processed.

Content and conditions:

- \* The record has a Record Type number of 1025 and the Critical Bit MAY be set.

- \* In a PTP Key Response message, this record MUST be contained exactly once.
- \* The Record Body is defined as a set of records and MAY contain the following records:

NTS Record Name	Communication Type	Use	Reference
Security Associations (one or more)	Multicast / Unicast	mandatory	This document, Section 3.2.11
Validity Period	Multicast / Unicast	mandatory	This document, Section 3.2.18
PTP Time Server	Unicast	mandatory	This document, Section 3.2.10
Ticket	Unicast	mandatory	This document, Section 3.2.15

Table 14: Current Parameters container for PTP Key Response message

- \* The records Security Association and Validity Period MUST be contained exactly once.
- \* Additionally, the records PTP Time Server and Ticket MUST be included exactly once if the client wants a unicast connection and MUST NOT be included if the client wants to join a multicast group.
- \* In a PTP Registration Response message, the Current Parameters container record MUST be contained exactly once.
- \* The Record Body MUST contain the following records exactly:
- \* In a PTP Registration Response message, the Current Parameters Container record MUST be contained exactly once.
- \* The record body MAY contain the following records:

NTS Record Name	Use	Reference
AEAD Algorithm Negotiation	mandatory	This document, Section 3.2.1
Validity Period	mandatory	This document, Section 3.2.18
Ticket Key ID	mandatory	This document, Section 3.2.17
Ticket Key	mandatory	This document, Section 3.2.16

Table 15: Current Parameters container for PTP  
Registration Response Message

#### 3.2.4. End of Message

Used in NTS-KE and NTS-TSR protocol

The End of Message record is defined in IETF RFC8915 [RFC8915], Section 4:

"The record sequence in an NTS message SHALL be terminated by an "End of Message" record. The requirement that all NTS-KE messages be terminated by an End of Message record makes them self-delimiting."

Content and conditions:

- \* The record has a Record Type number of 0 and a zero-length body.
- \* The Critical Bit MUST be set.
- \* This record MUST occur exactly once as the final record of every NTS request and response message.
- \* This record SHOULD NOT be included in the container records and MUST be ignored if present.
- \* See also: IETF RFC8915, Section 4.1.1

#### 3.2.5. Error

Used in NTS-KE and NTS-TSR protocol

The Error record is defined in IETF RFC8915 [RFC8915], Section 4.1.3. In addition to the Error codes 0 to 2 specified there the following Error codes 3 to 4 are defined:

Error Code	Description
0	Unrecognized Critical Record
1	Bad Request
2	Internal Server Error
3	Not Authorized
4	Grantor not Registered
5 - 32767	Unassigned
32768 - 65535	Reserved for Private or Experimental Use

Table 16: Error Codes

Content and conditions:

- \* The record has a Record Type number of 2 and body length of two octets consisting of an unsigned 16-bit integer in network byte order, denoting an error code.
- \* The Critical Bit MUST be set.
- \* The Error code 3 "Not Authorized" is sent by the NTS-KE server if the requester is not authorized to join the desired multicast group or if a grantor is prohibited to register with the NTS-KE server.
- \* The Error record MUST NOT be included in a PTP Registration Request message.
- \* The Error code 4 "Grantor not Registered" is sent by the NTS-KE server when the requester wants to establish a unicast connection to a grantor that is not registered with the NTS-KE server.
- \* The Error record MUST NOT be included in a PTP Key Request message.

### 3.2.6. Heartbeat Timeout

Used in NTS-TSR protocol

This record provides the NTS-KE server the capability to monitor the availability of the registered grantors. If this optional record is used, the registered grantors SHOULD send an NTS Heartbeat message to the NTS-KE server before the timeout expires.

Content and conditions:

- \* The record has a Record Type number of 1026 and the Critical Bit SHOULD NOT be set.
- \* The Record Body consists of a 16-bit unsigned integer in network byte order and denotes the heartbeat timeout in seconds..
- \* The timeout set by the NTS-KE server MUST NOT be less than 1s and MUST be less than the lifetime set in the Validity Period record.
- \* The timeout starts at the NTS-KE server with the generation of the Registration Response message.
- \* Grantors that receive an invalid value as a heartbeat timeout MUST ignore this record and MUST NOT send heartbeat messages.
- \* Grantors that receive a valid value SHOULD send a heartbeat message to the NTS-KE server before the timeout has elapsed.
- \* The grantors SHOULD keep the heartbeat intervals and MAY also send heartbeat messages more frequently.
- \* After transmitting a heartbeat from the grantor to the NTS-KE server, both sides reset the timeout to the start value and let the time count down again.
- \* If this timeout is exceeded without receiving a heartbeat message or several heartbeats are missing in a row, the NTS-KE server MAY delete the grantor from its registration list, so that a new registration of the grantor is necessary.
- \* Grantors that are not (or no longer) registered with a NTS-KE server MUST NOT send heartbeat messages and NTS-KE servers MUST discard heartbeat messages from non-registered grantors.
- \* NTS-KE servers MAY respond in such cases with a Registration Response message containing error code 4 "Grantor not Registered".

### 3.2.7. Next Parameters

Used in NTS-KE and NTS-TSR protocol

This record is a simple container that can carry an arbitrary number of NTS records. It holds all security parameters relevant for the upcoming validity period. The content as well as further conditions are defined by the respective NTS messages. The order of the included records is arbitrary and the parsing rules are so far identical with the NTS message. One exception: An End of Message record SHOULD NOT be present and MUST be ignored. When the parser reaches the end of the Record Body quantified by the Body Length, all embedded records have been processed.

Content and conditions:

- \* The record has a Record Type number of 1027 and the Critical Bit MAY be set.
- \* The Record Body is defined as a set of records.

- \* The structure of the record body and all conditions MUST be identical to the rules described in Section 3.2.3 of this document.
- \* In both the PTP Key Response and PTP Registration Response message, this record MUST be contained exactly once during the update period.
- \* Outside the update period, this record MUST NOT be included.
- \* In GrBA mode, this record MAY also be missing if the requesting client is to be explicitly excluded from a multicast group after the security parameter rotation process by the NTS-KE server.
- \* More details are described in Section 2.2.1.

### 3.2.8. NTS Next Protocol Negotiation

Used in NTS-KE protocol

The Next Protocol Negotiation record is defined in IETF RFC8915 [RFC8915], Section 4.1.2:

\_"The Protocol IDs listed in the client's NTS Next Protocol Negotiation record denote those protocols that the client wishes to speak using the key material established through this NTS-KE server session. Protocol IDs listed in the NTS-KE server's response MUST comprise a subset of those listed in the request and denote those protocols that the NTP server is willing and able to speak using the key material established through this NTS-KE server session. The client MAY proceed with one or more of them. The request MUST list at least one protocol, but the response MAY be empty."\_

Content and conditions:

- \* The record has a Record Type number of 1 and the Critical Bit MUST be set.
- \* The Record Body consists of a sequence of 16-bit unsigned integers in network byte order.  
\*Record body = {Protocol ID 1 || Protocol ID 2 || ...}\*
- \* Each integer represents a Protocol ID from the IANA "Network Time Security Next Protocols" registry as shown in the table below.
- \* For NTS request messages for PTPv2.1 (NTS-KE protocol merely), only the Protocol ID for PTPv2.1 SHOULD be included.
- \* This prevents the mixing of records for different time protocols.

Protocol ID	Protocol Name	Reference
0	Network Time Protocol version 4 (NTPv4)	[RFC8915], Section 7.7
1	Precision Time Protocol version 2.1 (PTPv2.1)	This document
2 - 32767	Unassigned	
32768 - 65535	Reserved for Private or Experimental Use	

Table 17: NTS next protocol IDs

Possible NTP/PTP conflict:

- \* The support of multiple protocols in this record may lead to the problem that records in NTS messages can no longer be assigned to a specific time protocol.
- \* For example, an NTS request could include records for both NTP and PTP.
- \* However, NTS for NTP does not use NTS message types and the End of Message record is also not defined for the case of multiple NTS requests in one TLS message.
- \* This leads to the mixing of the records in the NTS messages.
- \* A countermeasure is the use of only a single time protocol in the NTS Next Protocol Negotiation record that explicitly assigns the NTS message to a specific time protocol.
- \* When using NTS-secured NTP and NTS-secured PTP, two separate NTS requests i.e., two separate TLS sessions MUST be made.

### 3.2.9. NTS Message Type

Used in NTS-TSR protocol

This record enables the distinction between different NTS message types and message versions for the NTS-TSR protocol. It MUST be included exactly once in each NTS message in the NTS-TSR protocol.

Content and conditions:

- \* The record has a Record Type number of 1028 and the Critical Bit MUST be set.
- \* The Record Body MUST consist of three data fields:

Field		Octets	Offset
Message Type		2	0
Message Version	Major version	1	2
Message Version (cont.)	Minor version	1	3

Table 18: Content of the NTS Message Type record

- \* The Message Type field is a 16-bit unsigned integer in network byte order, denoting the type of the current NTS message.
- \* The following values are defined for the Message Type:

Message Type (value)	NTS Message (NTS-TSR protocol)
0	PTP Registration Request
1	PTP Registration Response
2	PTP Registration Revoke
3	Heartbeat
4 - 32767	Unassigned
32768 - 65535	Reserved for Private or Experimental Use

Table 19: NTS Message Types for the NTS-TSR protocol

- \* The Message Version consists of a tuple of two 8-bit unsigned integers in network byte order:  
\*NTS Message Version = {major version || minor version}\*
  - \* The representable version is therefore in the range 0.0 to 255.255 (e.g., v1.4 = 0104h).
  - \* All NTS messages for PTPv2.1 described in this document are in version number 1.0.
  - \* Thus the Message Version MUST match 0100h.

### 3.2.10. PTP Time Server

Used in NTS-KE and NTS-TSR protocol

The PTP Time Server record is used exclusively in TiBA mode (PTP unicast connection) and signals to the client (PTP requester) for which grantor the security parameters are valid. This record is used both, in the NTS-KE protocol in the PTP Key Response, and in NTS-TSR protocol in the PTP Registration Request message.

Content and conditions:

- \* The record has a Record Type number of 1029 and the Critical Bit MAY be set.
- \* The record body consists of a tuple of two 8-bit unsigned integers in network byte order.
- \* The structure of the record body and all conditions MUST be identical to the rules described in Section 3.2.2 (Association Mode) of this document, with the following exceptions:
- \* In a PTP Key Response message, this record MUST be contained exactly once within a container record (e.g., Current Parameters container record).
- \* The PTP Time Server record contains a list of all available addresses of the grantor assigned by the NTS-KE server.
- \* This can be an IPv4, IPv6, MAC address, as well as the PortIdentity of the grantor.
- \* This allows the client to change the PTP transport mode (e.g., from IPv4 to 802.3) without performing a new NTS request.
- \* The list in the PTP Time Server record MUST NOT contain the Association Type number 0 (multicast group) and MUST contain at least one entry.
- \* The NTS-KE server SHOULD provide the grantor addresses requested by the client in the PTP Key Request message, but MAY also assign a different grantor to the client.
- \* In a PTP Registration Request message, this record MUST be included exactly once.
- \* The grantor MUST enter all network addresses that are supported for a unicast connection.
- \* This can be an IPv4, IPv6, MAC address, as well as the PortIdentity.
- \* The list in the PTP Time Server record MUST NOT contain the Association Type number 0 (multicast group) and MUST contain at least the PortIdentity.
- \* The PortIdentity is especially needed by the NTS-KE server to identify the correct PTP instance (the grantor) in case of a PTP Registration Revoke message.

### 3.2.11. Security Association

Used in NTS-KEprotocol

This record contains the information "how" specific PTP message types must be secured. It comprises all dynamic (negotiable) values necessary to construct the AUTHENTICATION TLV (IEEE Std 1588-2019, Section 16.14.3). Static values and flags, such as the secParamIndicator, are described in more detail in Section 6.

Content and conditions:

- \* The record has a Record Type number of 1030 and the Critical Bit MAY be set.
- \* The Record Body is a sequence of various parameters in network byte order and MUST be formatted according to the following table:

Field	Octets	Offset
Security Parameter Pointer	1	0
Integrity Algorithm Type	2	1
Key ID	4	3
Key Length	2	7
Key	K	9

Table 20: Security Association record

- \* In a PTP Key Response message, the Security Association record MUST be included exactly once in the Current Parameters container record and the Next Parameters container record.
- \* The Next Parameters container record MUST be present only during the update period.
- \* In TiBA mode, the Security Association record MUST be included exactly once in the encrypted Ticket as well.

#### Security Parameter Pointer

- \* The Security Parameter Pointer (SPP) is an 8-bit unsigned integer in the closed range 0 to 255.
- \* This value enables the mutual assignment of SA, SP and AUTHENTICATION TLVs.
- \* The generation and management of the SPP is controlled by the NTS-KE server (see Section 4.2).

#### Integrity Algorithm Type

- \* This value is a 16-bit unsigned integer in network byte order.
- \* The possible values are equivalent to the MAC algorithm types from the table in Section 3.2.14.
- \* The value used depends on the negotiated or predefined MAC algorithm.

#### Key ID

- \* The Key ID is a 32-bit unsigned integer in network byte order.
- \* The field length is oriented towards the structure of the AUTHENTICATION TLV.
- \* The generation and management of the Key ID is controlled by the NTS-KE server.
- \* The NTS-KE server MUST ensure that every Key ID is unique.
  - The value can be either a random number or an enumeration.
  - Previous Key IDs SHOULD NOT be reused for a certain number of rotation periods or a defined period of time (see Section 4.2).

#### Key Length

- \* This value is a 16-bit unsigned integer in network byte order, denoting the length of the key.

#### Key

- \* The value is a sequence of octets with a length of Key Length.
- \* This symmetric key is needed together with the MAC algorithm to calculate the ICV.
- \* It can be both a group key (GrBA mode) or a unicast key (TiBA mode).

### 3.2.12. Source PortIdentity

Used in NTS-KE and NTS-TSR protocol

This record contains a PTP PortIdentity and serves as an identifier. In a PTP Key Request message, it enables the unique assignment of the NTS request to the PTP instance of the sender, since the request may have been sent to the NTS-KE server via a management port.

The PortIdentity is embedded in the PTP Key Response message within the ticket to bind it to the PTP requester. Grantors can verify that the ticket comes from the correct sender when it is received and before it is decrypted, to prevent possible crypto-performance attacks. In a PTP registration Revoke message this record enables the assignment of the grantor at the NTS-KE server to revoke an existing registration. This is necessary because requesting PTP devices may have multiple independent PTP ports and possibly multiple registrations with the KE.

Content and conditions:

- \* The record has a Record Type number of 1031 and the Critical Bit MAY be set.
- \* The record contains the PTP PortIdentity of the sender in network byte order, with a total length of 10 octets.
- \* In a PTP Key Request message, this record MUST be included exactly once if the client intends a unicast request in TiBA mode and MUST NOT be included if the client intends to join a multicast group/Go2 (= GrBA mode).
- \* In a PTP Registration Revoke message, this record MUST be included exactly once.
- \* The PortIdentity consists of the attributes clockIdentity and portNumber:  
\*PortIdentity = {clockIdentity || portNumber}\*
- \* The clockIdentity is an 8-octet array and the portNumber is a 16-bit unsigned integer (source: [IEEE1588-2019], Sections 5.3.5 and 7.5)

### 3.2.13. Status

Used in NTS-TSR protocol

The Status record is an optional record that represents the current load of the sender. It allows the NTS-KE server to improve load balancing when assigning grantors to the requesting PTP clients in TiBA mode. The content of the record is designed in such a way that it can also transmit other information (e.g., manufacturer-related information).

Content and conditions:

- \* The record has a Record Type number of 1032 and the Critical Bit SHOULD NOT be set.
- \* The Record Body MUST consist of two data fields:

Field	Octets	Offset
Status Type	2	0
Status Data	D	2

Table 21: Structure of the Status record

- \* The Status Type is a 16-bit unsigned integer, denoting the content of the Status Data field.
- \* The Status Data field is a sequence of octets in network byte order whose length, content and structure is determined by the Status Type field.
- \* The following values are currently set:

Status Type	Status Data length	Description
0	1 octet (unsigned int)	grantor load
1 - 32767	Unassigned	
32767 - 65535	Reserved for Private or Experimental Use	

Table 22: Values for Status Data

- \* The following values apply to Status Type 0:

Status Type	Status Data value	Description
0	0x01	grantor load: 0% to 24%
0	0x02	grantor load: 25% to 49%
0	0x03	grantor load: 50% to 74%
0	0x04	grantor load: 75% to 84%
0	0x05	grantor load: 85% to 94%
0	0x06	grantor load: 95% to 100%

Table 23: Values for Status Type 0

- \* In a Heartbeat message this record MAY be contained once or several times.
- \* If multiple status records are included, the status type MUST NOT occur twice.
- \* The NTS-KE server MAY use the status record for optimizations and MAY also ignore them.

#### 3.2.14. Supported MAC Algorithms

Used in NTS-KE and NTS-TSR protocol

This record allows free negotiation of the MAC algorithm needed to generate the ICV. Since multicast groups are restricted to a shared algorithm, this record is used mandatorily in a PTP Registration Request message and MAY be used (optionally) in a PTP Key Request message.

Content and conditions:

- \* The record has a Record Type number of 1033 and the Critical Bit MAY be set.
- \* The Record Body contains a sequence of 16-bit unsigned integers in network byte order.  
\*Supported MAC Algorithms = {MAC 1 || MAC 2 || ...}\*
- \* Each integer represents a MAC Algorithm Type defined in the table below.
- \* Duplicate identifiers SHOULD NOT be included.
- \* Each PTP node MUST support at least the HMAC-SHA256-128 algorithm.

MAC Algorithm Types	MAC Algorithm	ICV Length (octets)	Reference
0	HMAC-SHA256-128	16	[fiPS-PUB-198-1], [IEEE1588-2019]
1	HMAC-SHA256	32	[fiPS-PUB-198-1]
2	AES-CMAC	16	[RFC4493]
3	AES-GMAC-128	16	[RFC4543]
4	AES-GMAC-192	24	[RFC4543]
5	AES-GMAC-256	32	[RFC4543]
6 - 32767	Unassigned		
32768 - 65535	Reserved for Private or Experimental Use		

Table 24: MAC Algorithms

In GrBA mode:

- \* This record is not necessary, since all PTP nodes in a multicast group MUST support the same MAC algorithm.
- \* Therefore, this record SHOULD NOT be included in a PTP Key Request message and the NTS-KE server MUST ignore this record if the Association Type in the Association Mode record is 0 (= multicast group).
- \* Unless this is specified otherwise by a PTP profile, the HMAC-SHA256-128 algorithm SHALL be used by default.

In TiBA mode:

- \* In a PTP Key Request message, this record MAY be contained if the requester wants a unicast connection (TiBA mode, not Go2) to a specific grantor.
- \* The requester MUST NOT send more than one record of this type.
- \* If this record is present, at least the HMAC-SHA256-128 MAC algorithm MUST be included.

- \* If multiple MAC algorithms are supported, the requester SHOULD put the desired algorithm identifiers in descending priority in the record body.
- \* Strong algorithms with higher bit lengths SHOULD have higher priority.
- \* The default MAC algorithm (HMAC-SHA256-128) MAY be omitted in the record.
- \* In a PTP Registration Request message, this record MUST be present and the grantor MUST include all supported MAC algorithms in any order.
- \* The NTS-KE server selects the algorithm after receiving a PTP Key Request message in unicast mode.
- \* The NTS-KE server SHOULD choose the highest priority MAC algorithm from the request message that grantor and requester support.
- \* The NTS-KE server MAY ignore the priority and choose a different algorithm that grantor and requester support.
- \* If the MAC Algorithm Negotiation record is not within the PTP Key Request message, the NTS-KE server MUST choose the default algorithm HMAC-SHA256-128.

#### Initialization Vector (IV)

- \* If GMAC is to be supported as a MAC algorithm, then an Initialization Vector (IV) must be constructed according to IETF RFC 4543 [RFC4543], Section 3.1.
- \* Therefore, the IV MUST be eight octets long and MUST NOT be repeated for a specific key.
- \* This can be achieved, for example, by using a counter.

#### 3.2.15. Ticket

Used in NTS-KE protocol

This record contains the parameters of the selected AEAD algorithm, as well as an encrypted security association. The record contains all the necessary security parameters that the grantor needs for a secured PTP unicast connection to the requester. The ticket is encrypted by the NTS-KE server with the symmetric ticket key which is also known to the grantor. The requester is not able to decrypt the encrypted security association within the ticket.

Content and conditions:

- \* The record has a Record Type number of 1034 and the Critical Bit MAY be set.
- \* The Record Body consists of several data fields and MUST be formatted as follows.

Field	Octets	Offset
Ticket Key ID	4	0
Source PortIdentity	10	4
Nonce Length	2	14
Nonce	N	16
Encrypted SA Length	2	N+16
Encrypted Security Association	E	N+18

Table 25: Structure of a Ticket record

- \* In a PTP Key Response message, this record MUST be included exactly once each in the Current Parameters container record and the Next Parameters container record if the requesting client wants a unicast communication to a specific grantor in TiBA mode.
- \* The Next Parameters container record MUST be present only during the update period.

## Ticket Key ID

- \* This is a 32-bit unsigned integer in network byte order, denoting the Key ID of the ticket key.
- \* The value is set by the and is valid for the respective validity period.
- \* See also Section 3.2.17 for more details.

## Source PortIdentity

- \* This 10-octet long field contains the identical Source PortIdentity of the PTP client from the PTP Key Request message.

## Nonce Length

- \* This is a 16-bit unsigned integer in network byte order, denoting the length of the Nonce field.

## Nonce

- \* This field contains the Nonce needed for the AEAD operation.
- \* The length and conditions attached to the Nonce depend on the AEAD algorithm used.

- \* More details and conditions are described in Section 4.1.

#### Encrypted SA Length

- \* This is a 16-bit unsigned integer in network byte order, denoting the length of the Encrypted Security Association field.

#### Encrypted Security Association

- \* This field contains the output of the AEAD operation ("Ciphertext") after the encryption process of the respective Record Body of the respective Security Association record.
- \* The plaintext of this field is described in Section 3.2.11.
- \* More details about the AEAD process and the required input data are described in Section 4.1.

### 3.2.16. Ticket Key

Used in NTS-TSR protocol

This record contains the ticket key, which together with an AEAD algorithm is used to encrypt and decrypt the ticket payload (content of the Encrypted Security Association field in the Ticket record).

Content and conditions:

- \* The record has a Record Type number of 1035 and the Critical Bit MAY be set.
- \* The Record Body consists of a sequence of octets holding the symmetric key for the AEAD function.
- \* The generation and length of the key MUST meet the requirement of the associated AEAD algorithm.
- \* In a PTP Registration Response message, this record MUST be included exactly once each in the Current Parameters container record and the Next Parameters container record.
- \* The Next Parameters container record MUST be present only during the update period.

### 3.2.17. Ticket Key ID

Used in NTS-TSR protocol

The Ticket Key ID record is a unique identifier that allows a grantor to identify the associated ticket key. The NTS-KE server is responsible for generating this Key ID, which is also unique to the PTP network and incremented at each rotation period. The associated key is known only to the NTS-KE server and grantor, and is generated and exchanged during the registration phase of the grantor. All tickets generated by the NTS-KE server for the corresponding grantor in this validity period using the same ticket key ID.

Content and conditions:

- \* The record has a Record Type number of 1036 and the Critical Bit MAY be set.
- \* The Record Body consists of a 32-bit unsigned integer in network byte order.
- \* The generation and management of the ticket key ID is controlled by the NTS-KE server.
- \* The NTS-KE server must ensure that every ticket key has a unique number.
  - The value is implementation dependent and MAY be either a random number, a hash value or an enumeration.
  - Previous IDs SHOULD NOT be reused for a certain number of rotation periods or a defined period of time.
- \* In a PTP Key Response message, this record MUST be included exactly once each in the Current Parameters container record and the Next Parameters container record if a unicast connection in TiBA mode is to be established.
- \* If the requester wishes to join a multicast group, the Ticket Key ID record MUST NOT be included in the container records.
- \* In a PTP Registration Response message, this record MUST be included exactly once in the Current Parameters container record and once in the Next Parameters container record.
- \* The Next Parameters container record MUST be present only during the update period.
- \* The Ticket record MUST be present in TiBA mode and MUST NOT be present in GrBA mode.

### 3.2.18. Validity Period

Used in NTS-KE and NTS-TSR protocol

This record contains the validity information of the respective security parameters see also Section 2.2.1).

Content and conditions:

- \* In a PTP Key Response as well as the PTP Registration Response message, this record MUST be included exactly once each in the Current Parameters container record and the Next Parameters container record.
- \* The record has a Record Type number of 1037 and the Critical Bit MAY be set.
- \* The Record Body MUST consist of three data fields:

Field	Octets	Offset
Lifetime	4	0
Update Period	4	4
Grace Period	4	8

Table 26: Structure of a  
Validity Period record

#### Lifetime

- \* The Lifetime is a 32-bit unsigned integer in network byte order.
- \* If this record is within a Current Parameters container record, it shows the remaining lifetime of the security parameters for the current validity period in seconds.
- \* If this record is within a Next Parameters container record, it shows the total lifetime of the security parameters for the next validity period in seconds.
- \* The counting down of the Next Parameters lifetime starts as soon as the remaining lifetime of the Current Parameters reaches 0s.
- \* The maximum value is set by the NTS-KE administrator or the PTP profile.
- \* In conjunction with a PTP unicast establishment in TiBA mode, the lifetime of the unicast key (within the Security Association record), the ticket key and registration lifetime of a grantor with the NTS-KE server MUST be identical.

#### Update Period

- \* The Update Period is a 32-bit unsigned integer in network byte order.
- \* It specifies how many seconds before the lifetime expires the update period starts.
- \* Unlike the lifetime, this is a fixed value that is not counted down.

- \* The Update Period value MUST NOT be greater than the full Lifetime.
- \* Recommended is an Update Period of 120s-300s if the full Lifetime is 900s or longer.
- \* If the value of the Update Period in the Current Parameters container record is greater than the Lifetime, then the key update process has started.
- \* The presence or absence of the Next Parameters container record is specified in Section 3.2.7.

#### Grace Period

- \* The Grace Period is a 32-bit unsigned integer in network byte order.
- \* It defines how many seconds expired security parameters MUST still be accepted.
- \* This allows the verification of incoming PTP messages that were still on the network and secured with the old parameters.
- \* The Grace Period value MUST NOT be greater than the Update Period.
- \* Recommended is a Grace Period of 5 to 10 seconds.

#### Notes:

- \* Requests during the currently running lifetime will receive respectively adapted count values.
- \* The lifetime is a counter that is decremented and marks the expiration of defined parameters when the value reaches zero.
- \* The realization is implementation-dependent and can be done for example by a secondly decrementing.
- \* It MUST be ensured that jumps (e.g., by adjustment of the local clock) are avoided.
- \* The use of a monotonic clock is suitable for this.
- \* Furthermore, it is to be considered which consequences the drifting of the local clock can cause.
- \* With sufficiently small values of the lifetime (<12 hours), this factor should be negligible.

## 4. Additional Mechanisms

This section provides information about the use of the negotiated AEAD algorithm as well as the generation of the security policy pointers.

### 4.1. AEAD Operation

General information about AEAD:

- \* The AEAD operation enables the integrity protection and the optional encryption of the given data, depending on the input parameters.
- \* While the structure of the AEAD output after the securing operation is determined by the negotiated AEAD algorithm, it usually contains an authentication tag in addition to the actual ciphertext.
- \* The authentication tag provides the integrity protection, whereas the ciphertext represents the encrypted data.
- \* The AEAD algorithms supported in this document (see Section 3.2.1) always return an authentication tag with a fixed length of 16 octets.
- \* The size of the following ciphertext is equal to the length of the plaintext.
- \* The concatenation of authentication tag and ciphertext always form the unit "Ciphertext":  
\*Ciphertext = {authentication tag || ciphertext}\*- \* Hint: The term "Ciphertext" is distinguished between upper and lower case letters.
- \* The following text always describes "Ciphertext".
- \* Separation of the information concatenated in Ciphertext is not necessary at any time.
- \* Six parameters are relevant for the execution of an AEAD operation:
  - AEAD (...): is the AEAD algorithm itself
  - A: Associated Data
  - N: Nonce
  - K: Key
  - P: Plaintext
  - C: Ciphertext
- \* The protection and encryption of the data is done as follows: C = AEAD (A, N, K, P)
- \* Therefore, the output of the AEAD function is the Ciphertext.
- \* The verification and decryption of the data is done this way: P = AEAD (A, N, K, C)
- \* The output of the AEAD function is the Plaintext if the integrity verification is successful.

AEAD algorithm and input/output values for the Ticket record:

- \* AEAD (...):
  - The AEAD algorithm that is negotiated between grantor and NTS-KE server during the registration phase.
  - A list of the AEAD algorithms considered in this document can be found in Section 3.2.1.
- \* Associated Data:

- The Associated Data is an optional AEAD parameter and can be of any length and content, as long as the AEAD algorithm does not give any further restrictions.
  - In addition to the Plaintext, this associated data is also included in the integrity protection.
  - When encrypting or decrypting the Security Association record, this parameter MUST remain empty.
- \* Nonce:
- Corresponds to the value from the Nonce field in the Ticket (Section 3.2.15).
  - The requirements and conditions depend on the selected AEAD algorithm.
  - For the AEAD algorithms defined in Section 3.2.1 (with numeric identifiers 15, 16, 17), a cryptographically secure random number MUST be used.
  - Due to the block length of the internal AES algorithm, the Nonce SHOULD have a length of 16 octets.
- \* Key:
- This is the symmetric key required by the AEAD algorithm.
  - The key length depends on the selected algorithm.
  - When encrypting or decrypting the Security Association record, the ticket key MUST be used.
- \* Plaintext:
- This parameter contains the data to be encrypted and secured.
  - For AEAD encryption, this corresponds to the Record Body of the Security Association record with all parameters inside.
  - This is also the output of the AEAD operation after the decryption process.
- \* Ciphertext:
- Corresponds to the value from the Encrypted Security Association field in the Ticket (Section 3.2.15).
  - The Ciphertext is the output of the AEAD operation after the encryption process.
  - This is also the input parameter for the AEAD decryption operation.

#### 4.2. SA/SP Management

This section describes the requirements and recommendations attached to SA/SP management, as well as details about the generation of identifiers.

Requirements for the Security Association Database management:

- \* The structure and management of the Security Association Database (SAD) are implementation-dependent both on the NTS-KE server and on the PTP devices.

- \* An example of this, as well as other recommendations, are described in Annex B.
- \* A PTP device MUST contain exactly one SAD and Security Policy Database (SPD).
- \* For multicast and Group-of-2 connections, SPPs MUST NOT occur more than once in the SAD of a PTP device.
- \* For unicast connections, SPPs MAY occur more than once in the SAD of a PTP device.
- \* The NTS-KE server MUST ensure that SPPs can be uniquely assigned to a multicast group or unicast connection.
- \* This concerns both the NTS-KE server and all PTP devices assigned to the NTS-KE server.

#### SPP generation:

The generation of the SPP always takes place on the NTS-KE server and enables the identification of a corresponding SA. The value of the SPP can be either a random number or an enumeration. An SPP used in any multicast group MUST NOT occur in any other multicast group or unicast connection. If a multicast group or unicast connection is removed by the NTS-KE server, the released SPPs MAY be reused for new groups or unicast connections. Before reusing an SPP, the NTS-KE server MUST ensure that the SPP is no longer in use in the PTP network (e.g., within Next Parameters). In different PTP devices, an SPP used in a unicast connection MAY also occur in another unicast connection, as long as they are not used in multicast groups.

#### Key/Key ID generation:

The generation of the keys MUST be performed by using a Cryptographically Secure Pseudo Random Number Generator (CSPRNG) on the NTS-KE server (see also Section 2.2.2). The length of the keys depends on the MAC algorithm used. The generation and management of the Key ID is also controlled by the NTS-KE server. The NTS-KE server MUST ensure that every Key ID is unique at least within an SA with multiple parameter sets. The value of the Key ID is implementation dependent and MAY be either a random number, a hash value or an enumeration. Key IDs of expired keys MAY be reused but SHOULD NOT be reused for a certain number of rotation periods or a defined period of time. Before reusing a Key ID, the NTS-KE server MUST be ensured that the Key ID is no longer in use in the PTP network (e.g., within Next Parameters).

## 5. New TICKET TLV for PTP Messages

Once a PTP port is registered as a grantor for association in unicast mode another PTP port (requester) can associate with it by first requesting a key from the NTS-KE server with Association Type in the Association Mode record set to one of the values 1 to 4 (IPv4, IPv6, 802.3 or PortIdentity), and Association Values to the related address of the desired grantor. After the reception of a PTP Key Response message during the NTS-KE protocol the requester obtains the unicast key and the Ticket record containing the Record Body of the Security Association record (see Section 2.1.2 and Section 3.2.15). The ticket includes the identification of the requester, the Encrypted SA along with the unicast key as well as the lifetime in the Validity record.

To provide the grantor with the security data, the requester sends a secured unicast request to the grantor, e.g., an Announce request (= Signaling message with a REQUEST\_UNICAST\_TRANSMISSION TLV with Announce as messageType in the TLV), which is secured with the unicast key.

To accomplish that, the requester sends a newly defined TICKET TLV with the Ticket embedded and the AUTHENTICATION TLV with the PTP unicast negotiation message. The TICKET TLV must be positioned before the AUTHENTICATION TLV to include the TICKET TLV in the securing by the ICV. The receiving grantor decrypts the Ticket (actually the encrypted security association) from the TICKET TLV getting access to the information therein. With the contained unicast key, the grantor checks the requester identity and the authenticity of the request message.

Thereafter, all secured unicast messages between grantor and requester will use the unicast key for generating the ICV in the AUTHENTICATION TLV for authentication of the message until the unicast key expires.

If the requester's identity does not match with the Source PortIdentity field in the Ticket or the ICV in the AUTHENTICATION TLV is not identical to the generated ICV by the grantor, then the unicast request message MUST be denied.

The TICKET TLV structure is given in Table 27 below.

field	Octets	Offset
tlvType	2	0
lengthfield	2	2
Ticket record	T	4

Table 27: Structure of the  
TICKET TLV

To comply with the TLV structure of IEEE Std 1588-2019 ([IEEE1588-2019], Section 14.1) the TICKET TLV is structured as presented in Table 27 with a newly defined tlvType, a respective length field and the Ticket record (see Section 3.2.15) containing the encrypted security association. Eventually the Ticket TLV may be defined externally to IEEE 1588 SA, e.g., by the IETF. Then the structure should follow IEEE Std 1588-2019 ([IEEE1588-2019], Section 14.3) to define a new standard organization extension TLV as presented in Table 28 below.

field	Octets	Offset
tlvType	2	0
lengthfield	2	2
organizationId	3	4
organizationSubType	3	7
Ticket record	T	10

Table 28: Structure of an  
organization extension TLV form for  
the TICKET TLV

The TICKET TLV will be added to the PTP message preceding the AUTHENTICATION TLV as shown in figure 48 of IEEE Std 1588-2019 ([IEEE1588-2019], Section 16.14.1.1).

## 6. AUTHENTICATION TLV Parameters

The AUTHENTICATION TLV is the heart of the integrated security mechanism (prong A) for PTP. It provides all necessary data for the processing of the security means. The structure is shown in Table 29 below (compare to figure 49 of [IEEE1588-2019]).

field	Use	Description
tlvType	mandatory	TLV Type
lengthfield	mandatory	TLV Length Information
SPP	mandatory	Security Parameter Pointer
secParamIndicator	mandatory	Security Parameter Indicator
keyID	mandatory	Key Identifier or Current Key Disclosure Interval, depending on verification scheme
disclosedKey	optional	Disclosed key from previous interval
sequenceNo	optional	Sequence number
RES	optional	Reserved
ICV	mandatory	ICV based on algorithm OID

Table 29: Structure of the AUTHENTICATION TLV

The tlvType is AUTHENTICATION and lengthfield gives the length of the TLV. When using the AUTHENTICATION TLV with NTS key management, the SPP and keyID will be provided by the NTS-KE server in the PTP Key Response message

The optional disclosedKey, sequenceNo, and RES fields are omitted. So all of the flags in the SecParamIndicator MUST be FALSE.

ICV field contains the integrity check value of the particular PTP message calculated using the integrity algorithm defined by the key management.

## 7. IANA Considerations

Considerations should be made ...

...

## 8. Security Considerations

...

## 9. Acknowledgements

The authors would like to thank ...

## 10. References

### 10.1. Normative References

[fiPS-PUB-198-1]

National Institute of Standards and Technology (NIST),  
"The Keyed-Hash Message Authentication Code (HMAC)",  
NIST fiPS PUB 198-1, 2008.

[IEEE1588-2019]

Institute of Electrical and Electronics Engineers - IEEE  
Standards Association, "IEEE Standard for a Precision  
Clock Synchronization Protocol for Networked Measurement  
and Control Systems", IEEE Standard 1588-2019, 2019.

[ITU-T\_X.509]

International Telecommunication Union (ITU), "Information  
technology Open systems interconnection The Directory:  
Public-key and attribute certificate frameworks", ITU-T  
Recommendation X.509 (2008), November 2008.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119,  
DOI 10.17487/RFC2119, March 1997,  
<<https://www.rfc-editor.org/info/rfc2119>>.

[RFC4493] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The  
AES-CMAC Algorithm", RFC 4493, DOI 10.17487/RFC4493, June  
2006, <<https://www.rfc-editor.org/info/rfc4493>>.

- [RFC4543] McGrew, D. and J. Viega, "The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH", RFC 4543, DOI 10.17487/RFC4543, May 2006, <<https://www.rfc-editor.org/info/rfc4543>>.
- [RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <<https://www.rfc-editor.org/info/rfc5116>>.
- [RFC5297] Harkins, D., "Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)", RFC 5297, DOI 10.17487/RFC5297, October 2008, <<https://www.rfc-editor.org/info/rfc5297>>.
- [RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<https://www.rfc-editor.org/info/rfc7301>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <<https://www.rfc-editor.org/info/rfc8915>>.

## 10.2. Informative References

- [Langer\_et\_al.\_2020] Langer, M., Heine, K., Sibold, D., and R. Bermbach, "A Network Time Security Based Automatic Key Management for PTPv2.1", 2020 IEEE 45th Conference on Local Computer Networks (LCN), Sydney, Australia, DOI 10.1109/LCN48667.2020.9314809, November 2020, <<https://ieeexplore.ieee.org/document/9314809>>.

Authors' Addresses

Martin Langer  
Ostfalia University of Applied Sciences  
Salzdahlumer Straße 46/48  
38302 Wolfenbüttel  
Germany  
Email: [mart.langer@ostfalia.de](mailto:mart.langer@ostfalia.de)

Rainer Bermbach  
Ostfalia University of Applied Sciences  
Salzdahlumer Straße 46/48  
38302 Wolfenbüttel  
Germany  
Email: [r.bermbach@ostfalia.de](mailto:r.bermbach@ostfalia.de)

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: 19 August 2022

M. Lichvar  
Red Hat  
15 February 2022

Network Time Protocol Version 5  
draft-mlichvar-ntp-ntp5-04

Abstract

This document describes the version 5 of the Network Time Protocol (NTP).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 19 August 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Requirements Language . . . . .	3
2. Basic Concepts . . . . .	3
3. Data Types . . . . .	4
4. Message Format . . . . .	5
5. Extension Fields . . . . .	9
5.1. Padding Extension Field . . . . .	10
5.2. MAC Extension Field . . . . .	10
5.3. Reference IDs Request and Response Extension Fields . . .	10
5.4. Server Information Extension Field . . . . .	12
5.5. Correction Extension Field . . . . .	12
5.6. Reference Timestamp Extension Field . . . . .	15
5.7. Monotonic Timestamp Extension Field . . . . .	15
6. Measurement Modes . . . . .	16
7. Client Operation . . . . .	18
8. Server Operation . . . . .	20
9. NTPv5 Negotiation in NTPv4 . . . . .	22
10. Acknowledgements . . . . .	23
11. IANA Considerations . . . . .	23
12. Security Considerations . . . . .	23
13. References . . . . .	23
13.1. Normative References . . . . .	23
13.2. Informative References . . . . .	23
Author's Address . . . . .	24

## 1. Introduction

Network Time Protocol (NTP) is a protocol which enables computers to synchronize their clocks over network. Time is distributed from primary time servers to clients, which can be servers for other clients, and so on. Clients can use multiple servers simultaneously.

NTPv5 is similar to NTPv4 [RFC5905]. The main differences are:

1. The protocol specification (this document) describes only the on-wire protocol. Filtering of measurements, security mechanisms, source selection, clock control, and other algorithms, are out of scope.
2. For security reasons, NTPv5 drops support for the symmetric active, symmetric passive, broadcast, control, and private modes. The symmetric and broadcast modes are vulnerable to replay attacks. The control and private modes can be exploited for denial-of-service traffic amplification attacks. Only the client and server modes remain in NTPv5.

3. Timestamps are clearly separated from values used as cookies.
4. NTPv5 messages can be extended only with extension fields. The MAC field is wrapped in an extension field.
5. Extension fields can be of any length, even indivisible by 4, but are padded to a multiple of 4 octets. Extension fields specified for NTPv4 are compatible with NTPv5.
6. NTPv5 adds support for other timescales than UTC.
7. The NTP era number is exchanged in the protocol, which extends the unambiguous interval of the client from 136 years to about 35000 years.
8. NTPv5 adds a new measurement mode to provide clients with more accurate transmit timestamps.
9. NTPv5 works with sets of reference IDs to prevent synchronization loops over multiple hosts.
10. Resolution of the root delay and root dispersion fields is improved.
11. Clients don't leak information about their clock (e.g. timestamps).

#### 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

#### 2. Basic Concepts

The distance to the reference time sources in the hierarchy of servers is called stratum. Primary time servers, which are synchronized to the reference clocks, are stratum 1, their clients are stratum 2, and so on.

Root delay measures the total delay on the path to the reference time source used by the primary time server. Each client on the path adds to the root delay the NTP delay measured to the server it considers best for synchronization. The delay includes network delays and any delays between timestamping of NTP messages and their actual reception and transmission. Half of the root delay estimates the maximum error of the clock due to asymmetries in the delay.

Root dispersion estimates the maximum error of the clock due to the instability of the clocks on the path and instability of NTP measurements. Each server on the path adds its own dispersion to the root dispersion. Different clock models can be used. In a simple model, the clock can have a constant dispersion rate, e.g. 15 ppm as used in NTPv4.

The sum of the root dispersion and half of the root delay is called root distance. It is the estimated maximum error of the clock, taking into account asymmetry in delay and stability of clocks and measurements.

Servers have randomly generated reference IDs to prevent synchronization loops.

### 3. Data Types

NTPv5 uses few different data types. They are all in the network order. Beside signed and unsigned integers, it has also the following fixed-point types:

#### time16

A 16-bit fixed-point type containing values in seconds. It has 1 signed integer bit (i.e. it is just the sign) and 15 fractional bits. The minimum value is the fraction  $-32767/32768$  (almost -1 second), the maximum value is  $32767/32768$  (almost 1 second), and the resolution is about 30 microseconds. The type has a special value of 0x8000, which indicates an unknown value.

#### time32

A 32-bit fixed-point type containing values in seconds. It has 4 unsigned integer bits and 28 fractional bits. The maximum value is 16 seconds and the resolution is about 3.7 nanoseconds. Note that this is different than the 32-bit time format in NTPv4.

#### timestamp64

A 64-bit fixed-point type containing timestamps. It has 32 signed integer bits and 32 fractional bits. It spans an interval of about 136 years and has a resolution of about 0.23 nanoseconds. It can be used in different timescales. In the UTC timescale it

is the number of SI seconds since 1 Jan 1972 plus 2272060800, excluding leap seconds. Timestamps in the TAI timescale are the same except they include leap seconds and extra 10 seconds for the original difference between TAI and UTC in 1972, when leap seconds were introduced. One interval covered by the type is called an NTP era. The era starting at the epoch is era number 0, the following era is number 1, and so on.

Some fields use a logarithmic scale, where an 8-bit signed integer represents the rounded  $\log_2$  value of seconds. For example, a  $\log_2$  value of 4 is 2 to the power of 4 (16 seconds), or a  $\log_2$  value of -2 is 2 to the power of -2 (0.25 seconds).

#### 4. Message Format

NTPv5 servers and clients exchange messages as UDP datagrams. Clients send requests to servers and servers send them back responses. The format of the UDP payload is shown in Figure 1.

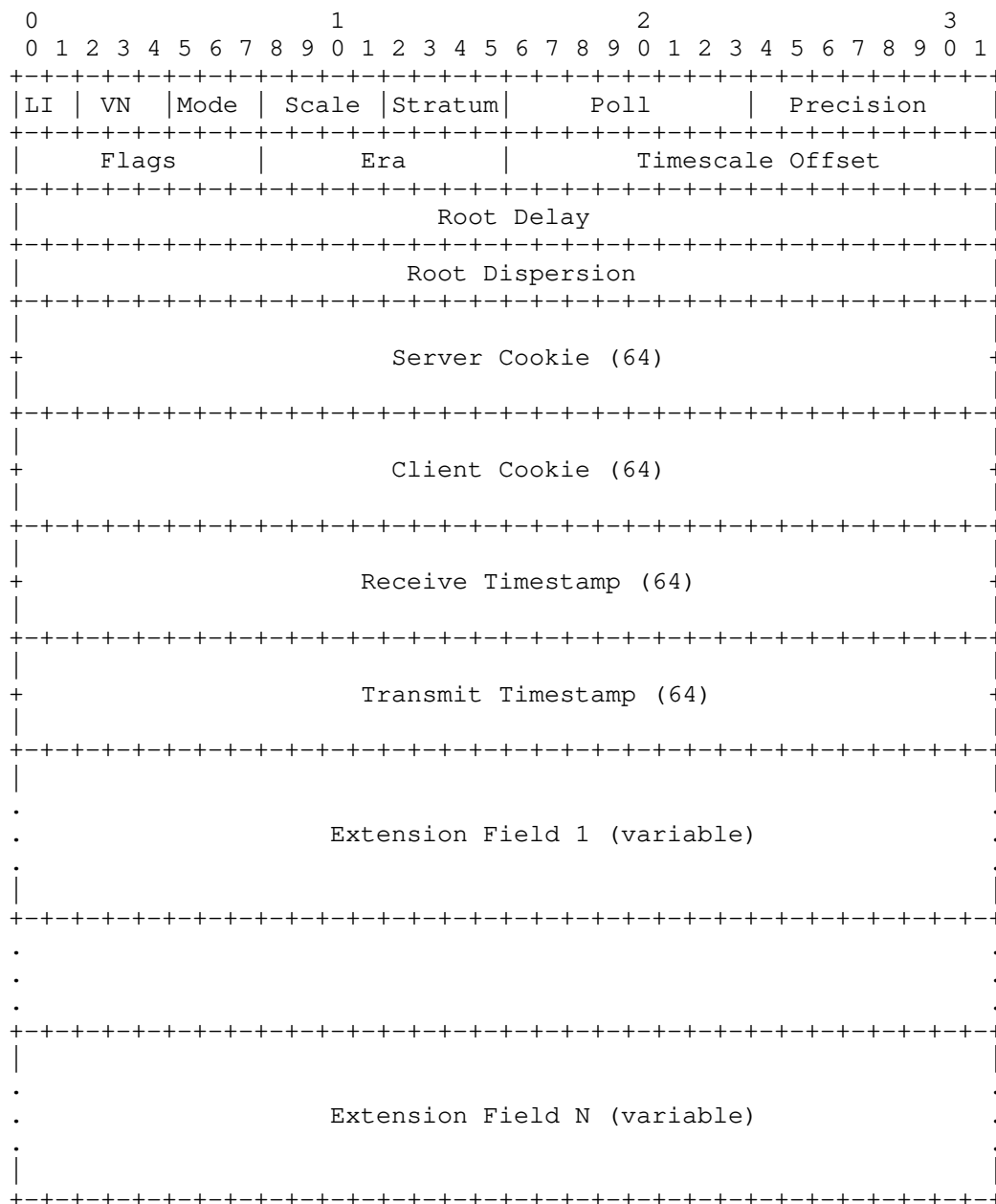


Figure 1: Format of NTPv5 messages

Each NTPv5 message has a header containing the following fields:

**Leap indicator (LI)**

A 2-bit field which can have the following values: 0 (normal), 1 (leap second inserted at the end of the month), 2 (leap second deleted at the end of the month), 3 (not synchronized). The values 1 and 2 are set at most 14 days in advance before the leap second. In requests it is always 0.

**Version Number (VN)**

A 3-bit field containing the value 5.

**Mode**

A 3-bit field containing the value 3 (request) or 4 (response).

**Scale**

A 4-bit identifier of the timescale. In requests it is the requested timescale. In responses it is the timescale of the receive and transmit timestamps. Defined values are:

0: UTC

1: TAI

2: UT1

3: Leap-smeared UTC

**Stratum**

A 4-bit field containing the stratum of host. Primary time servers have a stratum of 1, their clients have a stratum of 2, and so on. The value of 0 indicates an unknown or infinite stratum. In requests it is always 0.

**Poll**

An 8-bit signed integer containing the polling interval as a rounded log2 value in seconds. In requests it is the current polling interval. In responses it is the minimum allowed polling interval.

**Precision**

An 8-bit signed integer containing the precision of the timestamps included in the message as a rounded log2 value in seconds. In requests, which don't contain any timestamps, it is always 0.

**Flags**

An 8-bit integer that can contain the following flags:

**0x1: Unknown leap**

In requests it is zero. In responses it indicates the server does not have a time source which provides information about leap seconds and the client should interpret the Leap Indicator as having only two values: synchronized (0) and not synchronized (3).

**0x4: Interleaved mode**

In requests it is a request for a response in the interleaved mode. In responses it indicates the response is in the interleaved mode.

**Era**

An 8-bit unsigned NTP era number corresponding to the receive timestamp. In requests it is always 0.

**Timescale Offset**

A 16-bit value specific to the selected timescale, which is referenced to the receive timestamp. In requests it is always 0.

- \* In the UTC (0) and TAI (1) timescales it is the TAI-UTC offset (TAI minus UTC) as a signed integer, or 0x8000 if unknown.
- \* In the UT1 timescale (2) it is the UT1-UTC offset (UT1 minus UTC) using the timel6 type (0x8000 if unknown).
- \* In the leap-smear UTC (3), it is the current offset between the leap smeared time and UTC (former minus latter) using the timel6 type (0x8000 if unknown).

**Root Delay**

A field using the time32 type. In responses it is the server's root delay. In requests it is always 0.

**Root Dispersion**

A field using the time32 type. In responses it is the server's root dispersion. In requests it is always 0.

**Server Cookie**

A 64-bit field containing a number generated by the server which enables the interleaved mode. In requests it is 0, or a copy of the server cookie from the last response.

**Client Cookie**

A 64-bit field containing a random number generated by the client. Responses contain a copy of the field from the corresponding request, which allows the client to verify that the responses are valid responses to the requests.

**Receive Timestamp**

A field using the timestamp64 type. In requests it is always 0. In responses it is the time when the request was received. The timestamp corresponds to the end of the reception.

**Transmit Timestamp**

A field using the timestamp64 type. In requests it is always 0. In responses it is the beginning of the transmission of a response to the client. Which response it refers to depends on the selected mode (basic or interleaved). See Measurement Modes (Section 6) for detail.

The header has 48 octets, which is the minimum length of a valid NTPv5 message. A message can contain zero, one, or multiple extension fields. The maximum length is not specified, but the length is always divisible by 4.

**5. Extension Fields**

The format of NTPv5 extension fields is shown in Figure 2.

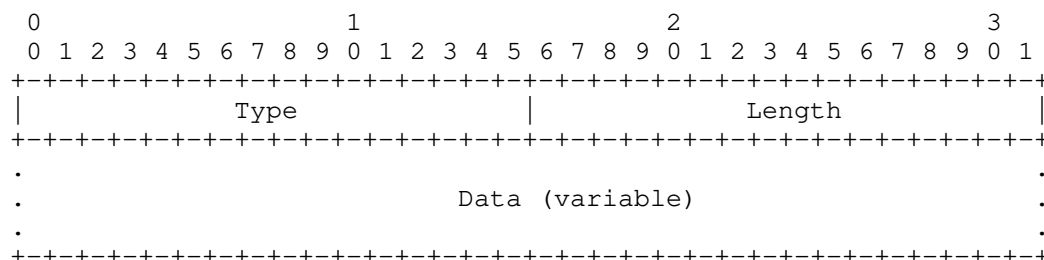


Figure 2: Format of NTPv5 extension fields

Each extension field has a header which contains a 16-bit type and 16-bit length. The length is in octets and it includes the header. The minimum length is 4, i.e. an extension field doesn't have to contain any data. If the length is not divisible by 4, the extension field is padded with zeroes to the smallest multiple of 4 octets.

If a request contains an extension field, the server **MUST** include this extension field in the response unless the specification of the extension field states otherwise, or the server does not support the extension field. A client can interpret the absence of an expected extension field in a response as an indication that the server does not support the extension field.

Extension fields specified for NTPv4 can be included in NTPv5 messages as specified for NTPv4.

The rest of this section describes new extension fields specified for NTPv5. Clients are not required to use or support any of these extension fields, but servers are required to support some extension fields.

#### 5.1. Padding Extension Field

This field is used by servers to pad the response to the same length as the request if the response doesn't contain all requested extension fields, or some have a variable length. It can have any length.

This field **MUST** be supported on server.

#### 5.2. MAC Extension Field

This field authenticates the NTPv5 message with a symmetric key. Implementations **SHOULD** use the MAC specified in RFC8573 [RFC8573]. The extension field **MUST** be the last extension field in the message unless an extension field is specifically allowed to be placed after a MAC or another authenticator field.

#### 5.3. Reference IDs Request and Response Extension Fields

Each NTPv5 server has a randomly generated 120-bit reference ID. The extension fields described in this section are used to exchange sets of reference IDs in order to detect synchronization loops, i.e. when a client is synchronizing (directly or indirectly) to one of its own clients.

As each client can be synchronized to an unlimited number of servers (and there can be up to 15 strata of servers), the reference IDs are exchanged as a Bloom filter instead of a list to limit the amount of data that needs to be exchanged.

The Bloom filter is an array of 4096 bits. When empty, all bits are zero. To add a reference ID to the filter, the 120-bit value of the reference ID is split into 10 12-bit values and the bits of the array at the 10 positions given by the 12-bit values are set to one.

A server maintains a copy of the filter for each server it is using as an NTP client. The filter provided by the server to clients is the union of the filters (using the bitwise OR operation) of the server's sources selected for synchronization and the server's own reference ID.

If the server uses a previous version of NTP for some of its sources, the reference IDs added to the filter are generated from their IP addresses as the first 120 bits of the MD5 sum of the address.

A client checking whether the server's set of reference IDs contains the client's own reference ID checks whether the bits at the 10 positions corresponding to the 12-bit values from the reference ID are all set to one. False positives are possible, but should be very rare for the specified length of the filter, even with a large number of reference IDs in the filter.

The filter can be exchanged as a single 512-octet array, or it can be exchanged in smaller chunks over multiple NTP messages, making them shorter, but delaying the detection of the synchronization loop.

The request extension field specifies the offset of the requested chunk in the filter as a number of octets. The requested length of the chunk is given by the length of the extension field. The response extension field **MUST** have the same length as the request extension field. If the request contains an invalid offset, the extension field **MUST** be ignored.

The client **SHOULD** use requests of a constant length for the association to avoid adding a variation to the measured NTP delay.

The format of the Reference IDs Request is shown in Figure 3.

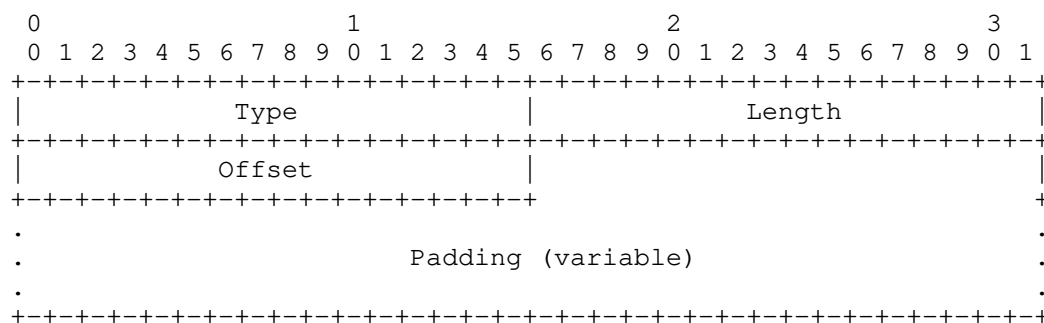


Figure 3: Format of Reference IDs Request Extension Field

The format of the Reference IDs Response is shown in Figure 4.

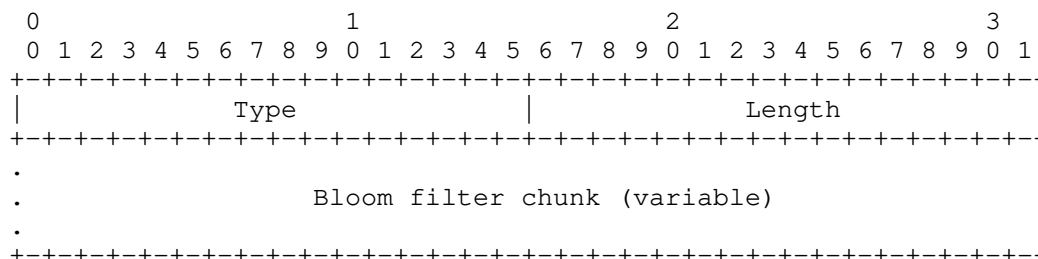


Figure 4: Format of Reference IDs Response Extension Field

These fields MUST be supported on server.

#### 5.4. Server Information Extension Field

This field provides clients with information about which NTP versions are supported by the server, as a minimum and maximum version. The extension field has a fixed length of 8 octets. In requests, all data fields of the extension are 0.

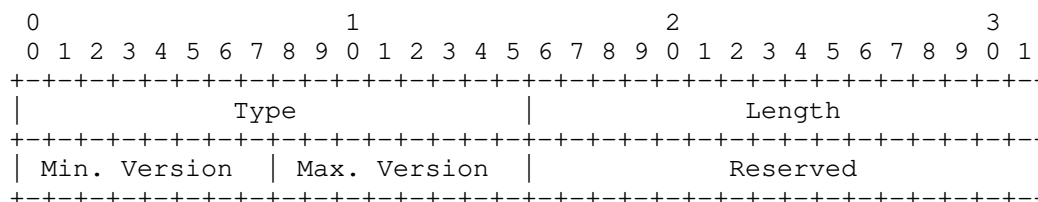


Figure 5: Format of Server Information Extension Field

This field MUST be supported on server.

#### 5.5. Correction Extension Field

Processing and queueing delays in network switches and routers may be a significant source of jitter and asymmetry in network delay, which has a negative impact on accuracy and stability of clocks synchronized by NTP. A solution to this problem is defined in the Precision Time Protocol (PTP) [IEEE1588], which is a different protocol for synchronization of clocks in networks. In PTP a special type of switch or router, called a Transparent Clock (TC), updates a correction field in PTP messages to account for the time messages spend in the TC. This is accomplished by timestamping the message at the ingress and egress ports, taking the difference to determine time in the TC and adding this to the Delay Correction. Clients can account for the accumulated Delay Correction to determine a more accurate clock offset.

The NTPv5 Delay Correction has the same format as the PTP correctionField to make it easier for manufacturers of switches and routers to implement NTP corrections. The format of the Correction Extension Field is shown in Figure 6.

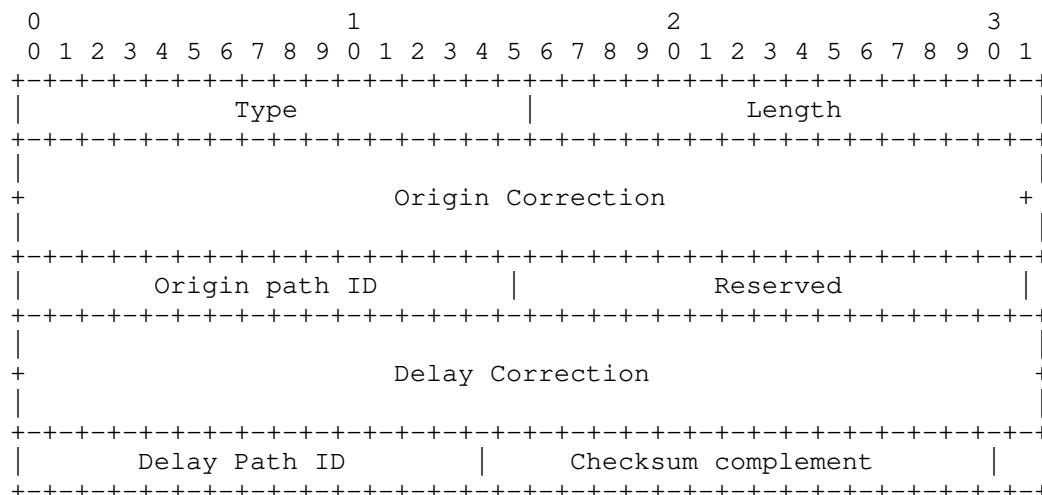


Figure 6: Format of Correction Extension Field

#### Field Type

The type which identifies the Correction extension field (value TBD).

#### Length

The length of the extension field, which is 28 octets.

#### Origin Correction

A field which contains a copy of the accumulated delay correction from the request packet in the NTP exchange.

#### Origin ID

A field which contains a copy of the final path ID from the request packet in the NTP exchange.

#### Reserved

16 bit reserved for future specification by the IETF. Transmit with all zeros.

#### Delay Correction

A signed fixed-point number of nanoseconds with 48 integer bits and 16 binary fractional bits, which represents the current correction of the network delay that has accumulated for this packet on the path from the source to the destination. The format of this field is identical to the PTP correctionField.

#### Path ID

A 16-bit identification number of the path where the delay correction was updated.

#### Checksum Complement

A field which can be modified in order to keep the UDP checksum of the packet valid. This allows the UDP checksum to be transmitted before the Correction Field is received and modified. The same field is described in RFC 7821 [RFC7821].

A correction capable client SHALL transmit the request with the Origin Correction, Origin ID, Delay Correction and Path ID fields filled with all zeros.

Network nodes, such as switches and routers, that are NTP corrections capable SHALL add the difference between the beginning of an NTP message retransmission and the end of the message reception to the received Delay Correction value, and update this field. Note that this time difference might be negative, for example in a cut-through switch. If the packet is transmitted at the same speed as it was received and the length of the packet does not change (e.g. due to adding or removing a VLAN tag), the beginning and end of the interval may correspond to any point of the reception and transmission as long as it is consistent for all forwarded packets of the same length. If the transmission speed or length of the packet is different, the beginning and end of the interval SHOULD correspond to the end of the reception and beginning of the transmission respectively. Both timestamps MUST be based on the same clock. This clock does not need to be synchronized as long as the frequency is accurate enough such that resulting time difference estimation errors are acceptable to the precision required by the application.

If a network node updates the delay correction, it SHOULD also add the identification numbers of the incoming and outgoing port to the path ID. Path ID values can be used by clients to determine if the ntp request and response messages are likely to have traversed the same network path.

If a network node modified any field of the extension field, it MUST update the checksum complement field in order to keep the current UDP checksum valid, or update the UDP checksum itself.

The server SHALL write the received Delay Correction value in the origin correction field of the response message, and the received path ID value in the origin ID field. The server SHALL set the Delay Correction field and Path ID fields to all zeros

#### 5.6. Reference Timestamp Extension Field

This fields contains the time of the last update of the clock. It has a fixed length of 12 octets. In requests, the timestamp is always 0.

(Is this really needed? It was mostly unused in NTPv4.)

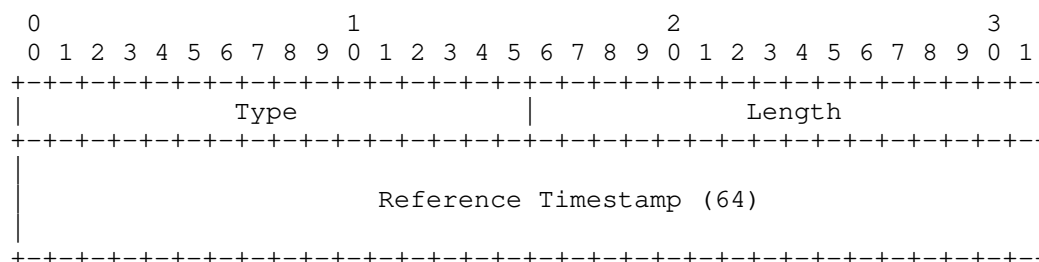


Figure 7: Format of Reference Timestamp Extension Field

#### 5.7. Monotonic Timestamp Extension Field

When a clock is synchronized to a time source, there is a compromise between time (phase) accuracy and frequency accuracy, because the frequency of the clock has to be adjusted to correct time errors that accumulate due to the frequency error (e.g. caused by changes in the temperature of the crystal). Faster corrections of time can minimize the time error, but increase the frequency error, which transfers to clients using that clock as a time source and increases their frequency and time errors. This issue can be avoided by transferring time and frequency separately using different clocks.

The Monotonic Timestamp Extension Field contains an extra receive timestamp with a 32-bit epoch identifier captured by a clock which doesn't have corrected phase and can better transfer frequency than the clock which captures the receive and transmit timestamps in the header. The extension field has a constant length of 16 octets. In requests, the counter and timestamp are always 0.

The epoch identifier is a random number which is changed when frequency transfer needs to be restarted, e.g. due to a step of the clock.

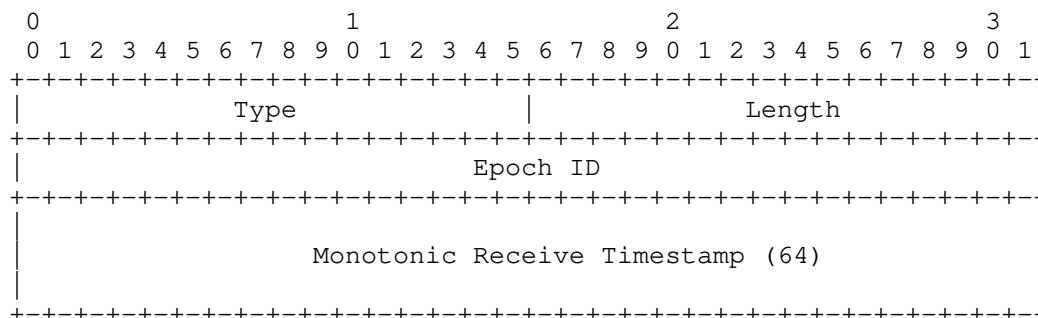


Figure 8: Format of Monotonic Timestamp Extension Field

The client can determine the frequency-transfer offset from the time-transfer offset and difference between the two receive timestamps in the response. It can use the frequency-transfer offset to better control the frequency of its clock, avoiding the frequency error in the server's time-transfer clock.

## 6. Measurement Modes

An NTPv5 client needs four timestamps to measure the offset and delay of its clock relative to the server's clock:

1. T1 - client's transmit timestamp of a request
2. T2 - server's receive timestamp of the request
3. T3 - server's transmit timestamp of a response
4. T4 - client's receive timestamp of the response

The offset, delay and dispersion are calculated as:

$$* \text{ offset} = ((T2 + T3) - (T4 + T1) + (Cd - Co)) / 2$$

$$* \text{ delay} = |(T4 - T1) - (T3 - T2) - (Cd + Co)|$$

$$* \text{ dispersion} = |T4 - T1| * DR$$

where

- \* T1, T2, T3, T4 are the receive and transmit timestamps of a request and response
- \* Co is the Origin Correction from the Correction Extension Field if present in the response and has acceptable values, zero otherwise

\* Cd is the Delay Correction from the Correction Extension Field if present in the response and has acceptable values, zero otherwise

\* DR is the client's dispersion rate

The client can make measurements in the basic mode, or interleaved mode if supported on the server. In the basic mode, the transmit timestamp in the server response corresponds to the message which contains the timestamp itself. In the interleaved mode it corresponds to a previous response identified by the server cookie. The interleaved mode enables the server to provide the client with a more accurate transmit timestamp which is available only after the previous response was formed or sent.

An example of cookies and timestamps in an NTPv5 exchange using the basic mode is shown in Figure 9.

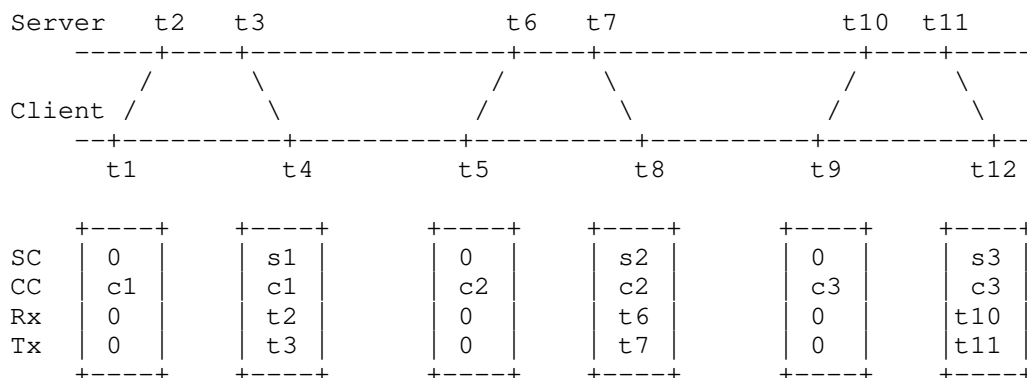


Figure 9: Cookies and timestamps in basic mode

From the three exchanges in this example, the client would use the the following sets of timestamps:

\* (t1, t2, t3, t4)

\* (t5, t6, t7, t8)

\* (t9, t10, t11, t12)

For NTPv4, the interleaved mode is described in NTP Interleaved Modes [I-D.ietf-ntp-interleaved-modes]. The difference between the NTPv5 and NTPv4 interleaved modes is that in NTPv5 it is enabled with a flag and the previous transmit timestamp on the server is identified by the server cookie instead of the receive timestamp.

An example of an NTPv5 exchange using the interleaved mode is shown in Figure 10. The messages in the basic and interleaved mode are indicated with B and I respectively. The timestamps  $t3'$  and  $t11'$  correspond to the same transmissions as  $t3$  and  $t11$ , but they may be less accurate. The first exchange is in the basic mode followed by a second exchange in the interleaved mode. For the third exchange, the client request is in the interleaved mode, but the server response is in the basic mode, because the server no longer had the timestamp  $t7$  (e.g. it was dropped to save timestamps for other clients using the interleaved mode).

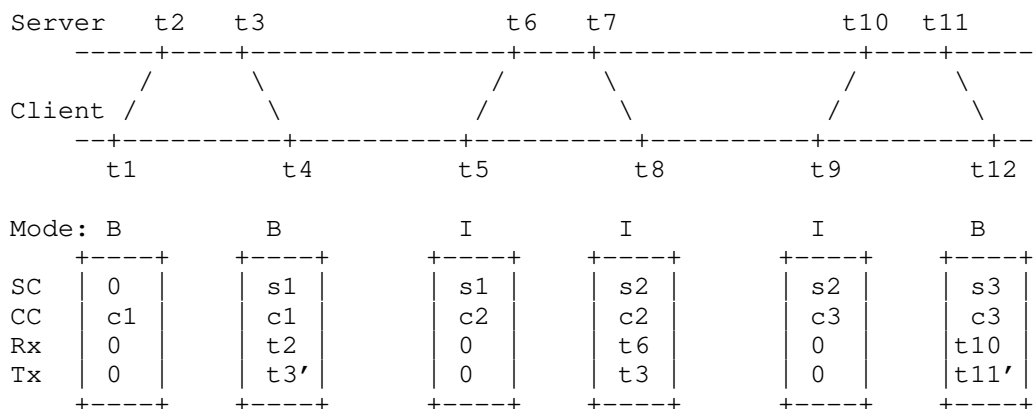


Figure 10: Cookies and timestamps in interleaved mode

From the three exchanges in this example, the client would use the following sets of timestamps:

- \* (t1, t2, t3', t4)
- \* (t1, t2, t3, t4) or (t5, t6, t3, t4)
- \* (t9, t10, t11', t12)

## 7. Client Operation

An NTPv5 client can use one or multiple servers. It has a separate association with each server. It makes periodic measurements of its offset and delay to the server. It can filter the measurements and compare measurements from different servers to select and combine the best servers for synchronization. It can adjust its clock in order to minimize its offset and keep the clock synchronized. These algorithms are not specified in this document.

The polling interval can be adjusted for the network conditions and stability of the clock. When polling a public server on Internet, the client SHOULD use at least a polling interval of 64 seconds, increasing up to at least 1024 seconds.

Each successful measurement provides the client with an offset, delay and dispersion. When combined with the server's root delay and dispersion, it gives the client an estimate of the maximum error.

On each poll, the client:

1. Generates a new random cookie.
2. Formats a request with necessary extension fields and the fields in the header all zero except:
  - \* Version is set to 5.
  - \* Mode is set to 3.
  - \* Scale is set to the timescale in which the client wants to operate.
  - \* Poll is set to the rounded log2 value of the current client's polling interval in seconds.
  - \* Flags are set according to the requested mode. The interleaved mode flag requests a response in the interleaved mode.
  - \* Server cookie is set only in the interleaved mode. If a valid response from the server was received previously, it is set to the server cookie from the previous response.
  - \* Client cookie is set to the newly generated cookie.
3. Sends the request to the server to the UDP port 123 and captures a transmit timestamp.
4. Waits for a valid response from the server and captures a receive timestamp. A valid response has version 5, mode 4, client cookie equal to the cookie from the request, and passes authentication if enabled. The client MUST ignore all invalid responses and accept at most one valid response.
5. Checks whether the response is usable for synchronization of the clock. Such a response has a leap indicator not equal to 3, stratum between 0 and 16, root delay and dispersion both smaller

than a specific value, e.g. 16 seconds, and timescale equal to the requested timescale. If the response is in a different timescale, the client can switch to the provided timescale, convert the timestamps if the offset between the timescales is provided or known, or drop the response.

6. Saves the server's receive and transmit timestamps. If the client internally counts seconds using a type wider than 32 bits, it SHOULD expand the timestamps with the provided NTP era.
7. Calculates the offset, delay, and dispersion.

A client which operates as a server for other clients MUST include the Reference IDs Request Extension Field in its requests in order to track reference IDs of its sources. If the server's set of reference IDs contains the client's own reference ID, it SHOULD not select the server for synchronization to avoid a synchronization loop.

## 8. Server Operation

A server receives requests on the UDP port 123. The server MUST support measurements in the basic mode. It MAY support the interleaved mode.

For the basic mode the server doesn't need to keep any client-specific state. For the interleaved mode it needs to save transmit timestamps and be able to identify them by a cookie.

The server maintains its leap indicator, stratum, root delay, and root dispersion:

- \* Leap indicator MUST be 3 if the clock is not synchronized or its maximum error cannot be estimated with the root delay and dispersion. Otherwise, it MUST be 0, 1, 2, depending on whether a leap second is pending in the next 14 day and, if it is, whether it will be inserted or deleted.
- \* Stratum SHOULD be one larger than stratum of the best server it uses for its own synchronization.
- \* Root delay SHOULD be the best server's root delay in addition to the measured delay to the server.
- \* Root dispersion SHOULD be the best server's root dispersion in addition to an estimate of the maximum drift of its own clock since the last update of the clock.

The server has a randomly generated 120-bit reference ID. It MUST track reference IDs of its servers in order to be able to respond with a Reference IDs Response Extension Field.

For each received request, the server:

1. Captures a receive timestamp.
2. Checks the version in the request. If it is not equal to 5, it MUST either drop the request, or handle it according to the specification corresponding to the protocol version. The server MAY respond with an NTPv5 message if and only if the request has version 5.
3. Drops the request if the format is not valid, mode is not 3, or authentication fails if the MAC Extension Field or another authenticator field is present. The server MUST ignore unknown extension fields.
4. Server forms a response with requested extension fields and sets the fields in the header as follows:
  - \* Leap Indicator, Stratum, Root delay, and Root dispersion, are set to the current server's values.
  - \* Version is set to 5.
  - \* Scale is set to the client's requested timescale if it is supported by the server. If not, the server SHOULD respond in any timescale it supports.
  - \* The flags are set as follows:
    - Unknown leap is set if the server does not know if a leap second is pending in the next 14 days, i.e. it has no source providing information about leap seconds.
    - Interleaved mode is set if the interleaved mode was requested and a response in the interleaved mode is possible (i.e. a transmit timestamp is associated with the server cookie).
  - \* Era is set to the NTP era of the receive timestamp.
  - \* Timescale Offset is set to the timescale-specific offset, or 0x8000 if unknown.

- \* Server Cookie is set when the interleaved mode is requested and it is supported by the server, even if the response cannot be in the requested mode due to the request having an unknown or invalid server cookie. The cookie identifies a more accurate transmit timestamp of the response, which can be retrieved by the client later with another request. The cookie generation is implementation-specific.
  - \* Client Cookie is set to the Client Cookie from the request.
  - \* Receive Timestamp is set to the server's receive timestamp of the request.
  - \* Transmit Timestamp is set to a value which depends on the measurement mode. In the basic mode it is the server's current time when the message is formed. In the interleaved mode it is the transmit timestamp of the previous response identified by the server cookie in the request, captured at some point after the message was formed.
5. Adds the Padding Extension field if necessary to make the length of the response equal to the length of the request.
  6. Drops the response if it is longer than the request to prevent traffic amplification.
  7. Sends the response.
  8. Saves the transmit timestamp and server cookie, if the interleaved mode was requested and is supported by the server.
9. NTPv5 Negotiation in NTPv4

NTPv5 messages are not compatible with NTPv4, even if they do not contain any extension fields. Some widely used NTPv4 implementations are known to ignore the version and interpret all requests as NTPv4. Their responses to NTPv5 requests have a zero client cookie, which means they fail the client's validation and are ignored.

The implementations are also known to not respond to requests with an unknown extension field, which prevents an NTPv4 extension field to be specified for NTPv5 negotiation. Instead, the reference timestamp field in the NTPv4 header is reused for this purpose.

An NTP server which supports both NTPv4 and NTPv5 SHOULD check the reference timestamp in all NTPv4 client requests. If the reference timestamp contains the value 0x4E5450354E545035 ("NTP5NTP5" in ASCII), it SHOULD respond with the same reference timestamp to indicate it supports NTPv5.

An NTP client which supports both NTPv4 and NTPv5, and is not configured to use a particular version, SHOULD start with NTPv4 requests having the reference timestamp set to 0x4e5450354e545035. If the server responds with the same reference timestamp, the client SHOULD switch to NTPv5.

## 10. Acknowledgements

Some ideas were taken from a different NTPv5 design proposed by Daniel Franke.

The author would like to thank Doug Arnold for his contributions and Dan Drown, Watson Ladd, Hal Murray, Kurt Roeckx, and Ulrich Windl for their suggestions and comments.

## 11. IANA Considerations

This memo includes no request to IANA.

## 12. Security Considerations

## 13. References

### 13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8573] Malhotra, A. and S. Goldberg, "Message Authentication Code for the Network Time Protocol", RFC 8573, DOI 10.17487/RFC8573, June 2019, <<https://www.rfc-editor.org/info/rfc8573>>.

### 13.2. Informative References

[I-D.ietf-ntp-interleaved-modes]

Lichvar, M. and A. Malhotra, "NTP Interleaved Modes", Work in Progress, Internet-Draft, draft-ietf-ntp-interleaved-modes-07, 18 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-ntp-interleaved-modes-07.txt>>.

[IEEE1588] Institute of Electrical and Electronics Engineers, "IEEE std. 1588-2019, "IEEE Standard for a Precision Clock Synchronization for Networked Measurement and Control Systems.", November 2019, <<https://www.ieee.org>>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.

[RFC7821] Mizrahi, T., "UDP Checksum Complement in the Network Time Protocol (NTP)", RFC 7821, DOI 10.17487/RFC7821, March 2016, <<https://www.rfc-editor.org/info/rfc7821>>.

#### Author's Address

Miroslav Lichvar  
Red Hat  
Purkynova 115  
612 00 Brno  
Czech Republic

Email: [mlichvar@redhat.com](mailto:mlichvar@redhat.com)

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: December 26, 2021

M. Lichvar  
Red Hat  
Jun 24, 2021

NTP Over PTP  
draft-mlichvar-ntp-over-ntp-00

Abstract

This document specifies a transport for the Network Time Protocol (NTP) client-server mode using the Precision Time Protocol (PTP) to enable hardware timestamping on hardware that can timestamp PTP messages but not NTP messages.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 26, 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## 1. Introduction

The Precision Time Protocol (PTP) [IEEE1588] was designed for highly accurate synchronization of clocks in a network. It relies on hardware timestamping supported in network devices (e.g. interface controllers, switches, and routers) to eliminate the impact of processing and queueing delays on PTP measurements.

PTP was originally designed for multicast communication. Later was added a unicast mode, which can be used in larger networks with partial on-path PTP support (e.g. telecom profiles G.8265.1 and G.8275.2).

The Network Time Protocol [RFC5905] does not rely on hardware timestamping support, but implementations can use it if it is available to avoid the impact of processing and queueing delays, similarly to PTP. The client-server mode of NTP is functionally similar to the PTP unicast mode.

An issue for NTP is hardware that can specifically timestamp only PTP packets. This limitation comes from their design, which does not allow the timestamps to be captured or retrieved at the same rate as packets can be received or transmitted. A filter needs to be implemented in the hardware to inspect each packet and timestamp only those that actually need it. The filter can be usually configured for the PTP transport (e.g. UDPv4, UDPv6, 802.3) and sometimes even the message type (e.g. sync message or delay request) to further reduce the rate of timestamps on the server or client side. This limitation prevents hardware timestamping of NTP messages. It also prevents timestamping of PTP messages if they are secured at the transport layer or below (e.g. IPSec or MACSec).

This document specifies a new transport for NTP to enable the PTP-specific timestamping support. It adds a new extension field (TLV) for PTP to contain NTP messages.

NTP over PTP does not disrupt normal operation of PTP. A network and even a single host can support both at the same time.

The specification does not take advantage of the PTP correctionField modified by PTP transparent clocks as their support for the unicast mode seems to be rare or nonexistent.

The client/server mode of NTP, even if using the PTP transport, has several advantages when compared to the PTP unicast mode:

- o It is more secure. It can use existing security mechanisms specified for NTP like Network Time Security [RFC8915], not losing

any of its features. The PTP unicast mode allows an almost-infinite traffic amplification, which can be exploited for denial-of-service attacks and can only be limited by security mechanisms using client authentication.

- o It needs fewer messages and less network bandwidth to get the same number of timestamps.
- o It is better suited for synchronization in networks without full on-path support. It does not assume the network delay is constant and the number of measurements in opposite directions is symmetric (in PTP sync messages and delay requests have independent timing).

### 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 2. PTP transport for NTP

A new TLV is defined for PTP to contain NTP messages in the client and server mode. Using other NTP modes in the TLV is not specified. Any transport specified for PTP that supports unicast messaging can be used for NTP over PTP, e.g. UDP on IPv4 and IPv6.

The type value of the NTP TLV is TBD. The TLV contains the whole NTP message as would normally be the UDP payload, without any modifications. The TLV does not propagate through boundary clocks.

If the UDP transport is used for PTP, the UDP source and destination port numbers MUST be the PTP event port (319). Client port randomization would break the timestamping.

The NTP TLV MUST be included in a delay request message. The originTimestamp field and all fields of the header SHOULD be zero, except:

- o messageType is 1 (delay request)
- o versionPTP is 2
- o messageLength is the length of the PTP message including the NTP TLV
- o domainNumber is TBD
- o flagField has the unicastFlag bit set

An NTP client using the PTP transport sends NTP requests in PTP messages to the server at the same rate as it would normally send them over UDP.

A server which supports the NTP TLV MUST check for the domainNumber of TBD and respond to an NTP request with a single PTP message containing the NTP response using the same PTP message format. It MUST NOT send a delay response message.

A server which does not support the NTP TLV will not recognize the domain number and ignore the message. If it responded to messages in the domain (e.g. due to misconfiguration), it would send a delay response (to port 320 if using the UDP transport), which would be ignored by the client.

Any authenticator fields included in the NTP messages MUST be calculated only over the NTP message following the header of the NTP TLV.

Timestamps SHOULD NOT be adjusted for the beginning of the NTP data in the PTP message. They SHOULD still correspond to the ending of the transmission and beginning of the reception (e.g. start of delimiter in the Ethernet frame).

Any modifications of the correctionField made by potential one-step end-to-end transparent clocks in the network SHOULD be ignored by the server and client.

### 3. Security Considerations

The PTP transport prevents NTP clients from randomizing their source port. It has no other impact on security of NTP.

### 4. References

#### 4.1. Normative References

- [IEEE1588] Institute of Electrical and Electronics Engineers, "IEEE std. 1588-2019, "IEEE Standard for a Precision Clock Synchronization for Networked Measurement and Control Systems.", 11 2019, <<https://www.ieee.org>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.

#### 4.2. Informative References

[RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <<https://www.rfc-editor.org/info/rfc8915>>.

#### Author's Address

Miroslav Lichvar  
Red Hat  
Purkynova 115  
Brno 612 00  
Czech Republic

Email: [mlichvar@redhat.com](mailto:mlichvar@redhat.com)

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: 22 April 2022

M. Lichvar  
Red Hat  
19 October 2021

NTP Over PTP  
draft-mlichvar-ntp-over-ntp-01

## Abstract

This document specifies a transport for the Network Time Protocol (NTP) client-server mode using the Precision Time Protocol (PTP) to enable hardware timestamping on hardware that can timestamp PTP messages but not NTP messages.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 22 April 2022.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. PTP transport for NTP . . . . .	3
3. Implementation Status - RFC EDITOR: REMOVE BEFORE PUBLICATION . . . . .	5
4. Security Considerations . . . . .	5
5. References . . . . .	5
5.1. Normative References . . . . .	6
5.2. Informative References . . . . .	6
Author's Address . . . . .	6

## 1. Introduction

The Precision Time Protocol (PTP) [IEEE1588] was designed for highly accurate synchronization of clocks in a network. It relies on hardware timestamping supported in network devices (e.g. interface controllers, switches, and routers) to eliminate the impact of processing and queueing delays on PTP measurements.

PTP was originally designed for multicast communication. Later was added a unicast mode, which can be used in larger networks with partial on-path PTP support (e.g. telecom profiles G.8265.1 and G.8275.2).

The Network Time Protocol [RFC5905] does not rely on hardware timestamping support, but implementations can use it if it is available to avoid the impact of processing and queueing delays, similarly to PTP. The client-server mode of NTP is functionally similar to the PTP unicast mode.

An issue for NTP is hardware that can specifically timestamp only PTP packets. This limitation comes from their design, which does not allow the timestamps to be captured or retrieved at the same rate as packets can be received or transmitted. A filter needs to be implemented in the hardware to inspect each packet and timestamp only those that actually need it. The filter can be usually configured for the PTP transport (e.g. UDPv4, UDPv6, 802.3) and sometimes even the message type (e.g. sync message or delay request) to further reduce the rate of timestamps on the server or client side. This limitation prevents hardware timestamping of NTP messages. It also prevents timestamping of PTP messages if they are secured at the transport layer or below (e.g. IPSec or MACSec).

This document specifies a new transport for NTP to enable the PTP-specific timestamping support. It adds a new extension field (TLV) for PTP to contain NTP messages.

NTP over PTP does not disrupt normal operation of PTP. A network and even a single host can support both at the same time.

The specification does not take advantage of the PTP correctionField modified by PTP transparent clocks as their support for the unicast mode seems to be rare or nonexistent.

The client/server mode of NTP, even if using the PTP transport, has several advantages when compared to the PTP unicast mode:

- \* It is more secure. It can use existing security mechanisms specified for NTP like Network Time Security [RFC8915], not losing any of its features. The PTP unicast mode allows an almost-infinite traffic amplification, which can be exploited for denial-of-service attacks and can only be limited by security mechanisms using client authentication.
- \* It needs fewer messages and less network bandwidth to get the same number of timestamps.
- \* It is better suited for synchronization in networks without full on-path support. It does not assume the network delay is constant and the number of measurements in opposite directions is symmetric (in PTP sync messages and delay requests have independent timing).

### 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 2. PTP transport for NTP

A new TLV is defined for PTP to contain NTP messages in the client, server, and symmetric modes. Using other NTP modes in the TLV is not specified. Any transport specified for PTP that supports unicast messaging can be used for NTP over PTP, e.g. UDP on IPv4 and IPv6.

The type value of the NTP TLV is TBD. The TLV contains the whole NTP message as would normally be the UDP payload, without any modifications. The TLV does not propagate through boundary clocks.

If the UDP transport is used for PTP, the UDP source and destination port numbers MUST be the PTP event port (319). Client port randomization would break the timestamping.

The NTP TLV MUST be included in a delay request message. The originTimestamp field and all fields of the header SHOULD be zero, except:

- \* messageType is 1 (delay request)
- \* versionPTP is 2
- \* messageLength is the length of the PTP message including the NTP TLV
- \* domainNumber is 123
- \* flagField has the unicastFlag (0x4) bit set

An NTP client using the PTP transport sends NTP requests in PTP messages to the server at the same rate as it would normally send them over UDP.

A server which supports the NTP TLV MUST check for the domainNumber of 123 and respond to an NTP request with a single PTP message containing the NTP response using the same PTP message format. It MUST NOT send a delay response message.

A server which does not support the NTP TLV will not recognize the domain number and ignore the message. If it responded to messages in the domain (e.g. due to misconfiguration), it would send a delay response (to port 320 if using the UDP transport), which would be ignored by the client.

Any authenticator fields included in the NTP messages MUST be calculated only over the NTP message following the header of the NTP TLV.

Timestamps SHOULD NOT be adjusted for the beginning of the NTP data in the PTP message. They SHOULD still correspond to the ending of the transmission and beginning of the reception (e.g. start of delimiter in the Ethernet frame).

Any modifications of the correctionField made by potential one-step end-to-end transparent clocks in the network SHOULD be ignored by the server and client.

### 3. Implementation Status - RFC EDITOR: REMOVE BEFORE PUBLICATION

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

#### 3.1. chrony

chrony (<https://chrony.tuxfamily.org>) has experimental support for PTP-over-NTP in its development branch. As the type of the NTP TLV, it uses 0x2023 from the experimental "do not propagate" range.

It was tested on Linux with the following network controllers, which have hardware timestamping limited to PTP packets:

Intel XL710 (i40e driver) - works

Intel X540-AT2 (ixgbe driver) - works

Intel 82576 (igb driver) - works

Broadcom BCM5720 (tg3 driver) - works

Broadcom BCM57810 (bnx2x driver) - does not timestamp unicast PTP packets

### 4. Security Considerations

The PTP transport prevents NTP clients from randomizing their source port. It has no other impact on security of NTP.

### 5. References

### 5.1. Normative References

- [IEEE1588] Institute of Electrical and Electronics Engineers, "IEEE std. 1588-2019, "IEEE Standard for a Precision Clock Synchronization for Networked Measurement and Control Systems."", November 2019, <<https://www.ieee.org>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.

### 5.2. Informative References

- [RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <<https://www.rfc-editor.org/info/rfc8915>>.

### Author's Address

Miroslav Lichvar  
Red Hat  
Purkynova 115  
612 00 Brno  
Czech Republic

Email: [mlichvar@redhat.com](mailto:mlichvar@redhat.com)

ntp  
Internet-Draft  
Intended status: Informational  
Expires: 7 August 2021

R. Salz  
Akamai Technologies  
3 February 2021

The update registries draft  
draft-rsalz-update-registries-02

## Abstract

The Network Time Protocol (NTP) and Network Time Security (NTS) documents define a number of assigned number registries, collectively called the NTP registries. Some registries have wrong values, some registries do not follow current common practice, and some are just right. For the sake of completeness, this document reviews all NTP and NTS registries.

## Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at <https://github.com/richsalz/draft-rsalz-update-registries>.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 7 August 2021.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Existing Registries . . . . .	3
2.1. Reference ID, Kiss-o'-Death . . . . .	3
2.2. Extension Field Types . . . . .	3
2.3. Network Time Security Registries . . . . .	4
3. New Registries . . . . .	4
4. IANA Considerations . . . . .	4
4.1. NTP Reference Identifier Codes . . . . .	4
4.2. NTP Kiss-o'-Death Codes . . . . .	5
4.3. NTP Extension Field Types . . . . .	5
4.4. Network Time Security Key Establishment Record Types . .	9
4.5. Network Time Security Next Protocols . . . . .	9
4.6. Network Time Security Error Codes . . . . .	9
4.7. Network Time Security Warning Codes . . . . .	9
5. Acknowledgements . . . . .	10
6. Normative References . . . . .	10
Author's Address . . . . .	11

## 1. Introduction

The Network Time Protocol (NTP) and Network Time Security (NTS) documents define a number of assigned number registries, collectively called the NTP registries. Some registries have wrong values, some registries do not follow current common practice, and some are just right. For the sake of completeness, this document reviews all NTP and NTS registries.

The bulk of this document can be divided into two parts:

- \* First, each registry, its defining document, and a summary of its syntax is defined.
- \* Second, the revised format and entries for each registry are defined.

## 2. Existing Registries

This section describes the registries and the rules for them. It is intended to be a short summary of the syntax and registration requirements for each registry. The semantics and protocol processing rules for each registry -- that is, how an implementation acts when sending or receiving any of the fields -- is not described here.

### 2.1. Reference ID, Kiss-o'-Death

[RFC5905] defined two registries, the Reference ID in Section 7.3, and the Kiss-o'-Death in Section 7.4. Both of these are allowed to be four ASCII characters; padded on the right with all-bits-zero if necessary. Entries that start with 0x58, the ASCII letter uppercase X, are reserved for private experimentation and development. Both registries are first-come first-served. The formal request to define the registries is in Section 16.

Section 7.5 of [RFC5905] defined the on-the-wire format of extension fields but did not create a registry for it.

### 2.2. Extension Field Types

[RFC5906] mentioned the Extension Field Types registry, and defined it indirectly by defining 30 extensions (15 each for request and response) in Section 13. It did not provide a formal definition of the columns in the registry. Section 10 of [RFC5906] splits the Field Type into four subfields, only for use within the Autokey extensions.

[RFC7821] added a new entry, Checksum Complement, to the Extension Field Types registry.

[RFC7822] clarified the processing rules for Extension Field Types, particularly around the interaction with the Message Authentication Code (MAC) field.

[RFC8573] changed the cryptography used in the MAC field.

The following problems exists with the current registry:

- \* Many of the entries in the Extension Field Types registry have swapped some of the nibbles; 0x1234 is listed as 0x1432 for example. This document marks the erroneous values as reserved.
- \* Some values were mistakenly re-used.

### 2.3. Network Time Security Registries

[RFC8915] defines the Network Time Security (NTS) protocol. Sections 7.1 through 7.5 (inclusive) added entries to existing registries.

Section 7.6 created a new registry, NTS Key Establishment Record Types, that partitions the assigned numbers into three different registration policies: IETF Review, Specification Required, and Private or Experimental Use.

Section 7.7 created a new registry, NTS Next Protocols, that similarly partitions the assigned numbers.

Section 7.8 created two new registries, NTS Error Codes and NTS Warning Codes. Both registries are also partitioned the same way.

### 3. New Registries

The following general guidelines apply to all registries defined here:

- \* Every entry reserves a partition for private use and experimentation.
- \* Registries with ASCII fields are now limited to uppercase letters; fields starting with 0x2D, the ASCII minus sign, are reserved for private use and experimentation.
- \* The policy for every registry is now specification required, as defined in Section 4.6 of [RFC8126].

The IESG is requested to choose three designated experts, with two being required to approve a registry change.

Each entry described in the below sub-sections is intended to completely replace the existing entry with the same name.

### 4. IANA Considerations

#### 4.1. NTP Reference Identifier Codes

The registration procedure is changed to specification required.

The Note is changed to read as follows:

- \* Codes beginning with the character "-" are reserved for experimentation and development. IANA cannot assign them.

The columns are defined as follows:

- \* ID (required): a four-byte value padded on the right with zero's. Each value must be an ASCII uppercase letter or minus sign
- \* Clock source (required): A brief text description of the ID
- \* Reference (required): the publication defining the ID.

The existing entries are left unchanged.

#### 4.2. NTP Kiss-o'-Death Codes

The registration procedure is changed to specification required.

The Note is changed to read as follows:

- \* Codes beginning with the character "-" are reserved for experimentation and development. IANA cannot assign them.

The columns are defined as follows:

- \* ID (required): a four-byte value padded on the right with zero's. Each value must be an ASCII uppercase letter or minus sign.
- \* Meaning source (required): A brief text description of the ID.
- \* Reference (required): the publication defining the ID.

The existing entries are left unchanged.

#### 4.3. NTP Extension Field Types

The registration procedure is changed to specification required.

The reference should be [RFC5906] added, if possible.

The following Note is added:

- \* Field Types in the range 0xF000 through 0xFFFF, inclusive, are reserved for experimentation and development. IANA cannot assign them. Both NTS Cookie and Autokey Message Request have the same Field Type; in practice this is not a problem as the field semantics will be determined by other parts of the message.

The columns are defined as follows:

- \* Field Type (required): A two-byte value in hexadecimal.

\* Meaning (required): A brief text description of the field type.

\* Reference (required): the publication defining the field type.

The table is replaced with the following entries.

Field Type	Meaning	Reference
0x0002	Reserved for historic reasons	This RFC
0x0102	Reserved for historic reasons	This RFC
0x0104	Unique Identifier	RFC 8915, Section 5.3
0x0200	No-Operation Request	RFC 5906
0x0201	Association Message Request	RFC 5906
0x0202	Certificate Message Request	RFC 5906
0x0203	Cookie Message Request	RFC 5906
0x0204	NTS Cookie	RFC 8915, Section 5.4
0x0204	Autokey Message Request	RFC 5906
0x0205	Leapseconds Message Request	RFC 5906
0x0206	Sign Message Request	RFC 5906
0x0207	IFF Identity Message Request	RFC 5906
0x0208	GQ Identity Message Request	RFC 5906
0x0209	MV Identity Message Request	RFC 5906
0x0302	Reserved for historic reasons	This RFC
0x0304	NTS Cookie Placeholder	RFC 8915, Section 5.5
0x0402	Reserved for historic reasons	This RFC
0x0404	NTS Authenticator and Encrypted Extension Fields	RFC 8915, Section 5.6

0x0502	Reserved for historic reasons	This RFC
0x0602	Reserved for historic reasons	This RFC
0x0702	Reserved for historic reasons	This RFC
0x2005	Reserved for historic reasons	This RFC
0x8002	Reserved for historic reasons	This RFC
0x8102	Reserved for historic reasons	This RFC
0x8200	No-Operation Response	RFC 5906
0x8201	Association Message Response	RFC 5906
0x8202	Certificate Message Response	RFC 5906
0x8203	Cookie Message Response	RFC 5906
0x8204	Autokey Message Response	RFC 5906
0x8205	Leapseconds Message Response	RFC 5906
0x8206	Sign Message Response	RFC 5906
0x8207	IFF Identity Message Response	RFC 5906
0x8208	GQ Identity Message Response	RFC 5906
0x8209	MV Identity Message Response	RFC 5906
0x8302	Reserved for historic reasons	This RFC
0x8402	Reserved for historic reasons	This RFC
0x8502	Reserved for historic reasons	This RFC
0x8602	Reserved for historic reasons	This RFC
0x8702	Reserved for historic reasons	This RFC
0x8802	Reserved for historic reasons	This RFC
0xC002	Reserved for historic reasons	This RFC
0xC102	Reserved for historic reasons	This RFC

0xC200	No-Operation Error Response	RFC 5906
0xC201	Association Message Error Response	RFC 5906
0xC202	Certificate Message Error Response	RFC 5906
0xC203	Cookie Message Error Response	RFC 5906
0xC204	Autokey Message Error Response	RFC 5906
0xC205	Leapseconds Message Error Response	RFC 5906
0xC206	Sign Message Error Response	RFC 5906
0xC207	IFF Identity Message Error Response	RFC 5906
0xC208	GQ Identity Message Error Response	RFC 5906
0xC209	MV Identity Message Error Response	RFC 5906
0xC302	Reserved for historic reasons	This RFC
0xC402	Reserved for historic reasons	This RFC
0xC502	Reserved for historic reasons	This RFC
0xC602	Reserved for historic reasons	This RFC
0xC702	Reserved for historic reasons	This RFC
0xC802	Reserved for historic reasons	This RFC
0x0902	Reserved for historic reasons	This RFC
0x8902	Reserved for historic reasons	This RFC
0xC902	Reserved for historic reasons	This RFC

Table 1

#### 4.4. Network Time Security Key Establishment Record Types

The registration procedure is changed to specification required.

The following note should be added:

- \* Record Type numbers in the range 0x4000 through 0x7FFF, inclusive, are reserved for experimentation and development. IANA cannot assign them.

The existing entries are left unchanged.

#### 4.5. Network Time Security Next Protocols

The registration procedure is changed to specification required.

The following note should be added:

- \* Protocol ID numbers in the range 0x8000 through 0xFFFF, inclusive, are reserved for experimentation and development. IANA cannot assign them.

The existing entries are left unchanged.

#### 4.6. Network Time Security Error Codes

The registration procedure is changed to specification required.

The following note should be added:

- \* Error code numbers in the range 0x8000 through 0xFFFF, inclusive, are reserved for experimentation and development. IANA cannot assign them.

The existing entries are left unchanged.

#### 4.7. Network Time Security Warning Codes

The registration procedure is changed to specification required.

The following note should be added:

- \* Warning code numbers in the range 0x8000 through 0xFFFF, inclusive, are reserved for experimentation and development. IANA cannot assign them.

The existing entries are left unchanged.

## 5. Acknowledgements

The members of the NTP Working Group helped a great deal. Notable contributors include.

- \* Miroslav Lichvar, RedHat
- \* Daniel Franke, Akamai Technologies
- \* Danny Mayer, Network Time Foundation
- \* Michelle Cotton, IANA

And thanks to Harlen Stenn for providing popcorn.

## 6. Normative References

- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.
- [RFC5906] Haberman, B., Ed. and D. Mills, "Network Time Protocol Version 4: Autokey Specification", RFC 5906, DOI 10.17487/RFC5906, June 2010, <<https://www.rfc-editor.org/info/rfc5906>>.
- [RFC7821] Mizrahi, T., "UDP Checksum Complement in the Network Time Protocol (NTP)", RFC 7821, DOI 10.17487/RFC7821, March 2016, <<https://www.rfc-editor.org/info/rfc7821>>.
- [RFC7822] Mizrahi, T. and D. Mayer, "Network Time Protocol Version 4 (NTPv4) Extension Fields", RFC 7822, DOI 10.17487/RFC7822, March 2016, <<https://www.rfc-editor.org/info/rfc7822>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8573] Malhotra, A. and S. Goldberg, "Message Authentication Code for the Network Time Protocol", RFC 8573, DOI 10.17487/RFC8573, June 2019, <<https://www.rfc-editor.org/info/rfc8573>>.

[RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <<https://www.rfc-editor.org/info/rfc8915>>.

Author's Address

Rich Salz  
Akamai Technologies

Email: [rsalz@akamai.com](mailto:rsalz@akamai.com)